Pragmatic Guide to JavaScript

JavaScript 修炼之道



[法] Christophe Porteneuve 著
巩朋 张铁 译

人民邮电出版社
POSTS & TELECOM PRESS

Christophe Porteneuve　从事IT研发十多年，并很早就专注于Web开发。2006年成为Prototype（http://prototypejs.org）的核心成员，2007年写作了*Prototype and script.aculo.us*一书。目前，他是法国Ciblo.net的CTO，并常在JavaScript的会议中做演讲。他和妻子Elodie现住在法国巴黎。

巩朋　本科毕业于大连理工大学，现为北京航空航天大学计算机系在读硕士。喜好读书，涉猎领域很广，尤其喜欢钻研程序设计语言理论。个人博客为http://www.cnblogs.com/figure9。

张铁　本科毕业于北京航空航天大学，现在该校计算机系攻读硕士。拥有丰富的Web前端开发经验和扎实的编程理论基础，对函数式程序设计语言有浓厚的兴趣。

图书在版编目（C I P）数据

JavaScript修炼之道 / (法) 波顿纽威
(Porteneuve,C.) 著 ; 巩朋, 张铁译. -- 北京 : 人民
邮电出版社, 2011.11
（图灵程序设计丛书）
书名原文: Pragmatic Guide to JavaScript
ISBN 978-7-115-26556-2

Ⅰ. ①J… Ⅱ. ①波… ②巩… ③张… Ⅲ. ①
JAVA语言—程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2011)第209827号

内 容 提 要

本书是 JavaScript 的实战秘籍。作者将自己多年的编程经验融入其中，不仅可以作为学习之用，更是日常 JavaScript 开发中不可多得的参考手册，使读者少走很多弯路。

本书的内容涵盖了当今流行的 JavaScript 库的运行机制，也提供了许多应用案例。本书针对各任务采取对页式编排，在对各任务的讲解中，左页解释了任务的实现原理，而右页则举出了该任务的代码片段以及可供对照参考的相关任务，便于读者阅读和理解。

本书的读者既包括 JavaScript 编程的新手，也包括已有不少 Web 应用编程经验的开发者。

图灵程序设计丛书
JavaScript修炼之道

◆ 著　[法] Christophe Porteneuve
译　巩　朋　张　铁
责任编辑　王军花
执行编辑　王一枝

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街14号
邮编　100061　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　　　　印刷

◆ 开本：800×1000　1/16
印张：8.25
字数：195千字　2011年 11 月第 1 版
印数：1 – 3 000册　2011年 11 月北京第 1 次印刷

著作权合同登记号　图字：01-2011-1376 号

ISBN 978-7-115-26556-2

定价：29.00元

读者服务热线：(010)51095186转604　印装质量热线：(010)67129223

反盗版热线：(010)67171154

版权声明

献　　词

谨以此书献给 Élodie——我最爱的妻子。

译　者　序

随着互联网的发展和移动上网设备数量的急剧增长，直接和用户打交道的网站前端变得越来越重要。而 JavaScript 作为前端开发的首选语言，其重要性不言而喻。

关于 JavaScript 的好书已有很多：入门级的有 Jeremy Keith 等人著的 *DOM Scripting*[①]，进阶级的有 Nicholas Zakas 的 *Professional JavaScript for Web Developers*[②]和 Douglas Crockford 的 *JavaScript: The Good Parts*[③]。不过，对于前端开发来说，仅仅学习 JavaScript 这门语言是远远不够的。正如本书作者所言，使用原始的 JavaScript 编写网站前端就像用石斧和原木来建造一座摩天大厦：即便构建一个简单的 Web 应用，DOM 中的不一致、浏览器兼容性问题以及 JavaScript 语言本身存在的问题也会使它变得诡异莫测。

本书作者从实用性出发，把前端开发中的常见问题归纳为一个个任务，给出了常用框架下实现这些任务的方法和注意事项。这些任务的覆盖面很广，从 JavaScript 语言的基本技巧，到 DOM 操纵、事件和定时器，再到 UI 技巧，甚至还包含现在流行的混搭（Mashup）技术。读者既可以把这本书作为 JavaScript 开发的快速参考手册，也可以通读本书，以深入理解 JavaScript 和各个流行框架的使用方法。

本书第一部分、第二部分和附录由巩朋翻译，其余部分则由张铁完成。我们已尽力保证译文准确、通顺，但限于自身的程序设计能力和文字表达水平，难免有所遗漏，希望读者在容忍的同时能够给予指正。

译　者

2011 年 8 月

---

① 此书中文版《JavaScript DOM 编程艺术（第 2 版）》已由人民邮电出版社出版。*（标*号的是译者注，下同。）

② 此书中文版《JavaScript 高级程序设计（第 2 版）》已由人民邮电出版社出版。*

③ 此书中文版《JavaScript 编程精粹》已由电子工业出版社出版。*

对本书的赞誉

“真希望我初学 JavaScript 那会就能有这本书！本书会让你在现实的 JavaScript 世界中游刃有余。它既讲述了当今流行的 JavaScript 库的运行原理，也提供了 JavaScript 正确实践的好建议和背景资料。作为最优秀的 JavaScript 开发者之一，作者将他多年的宝贵经验凝聚在本书中，使它成为日常 JavaScript 开发中的必读参考。”

——Thomas Fuchs

script.aculo.us 框架创始人

“这本书包含了一系列当今浏览器中既精妙又实用的 JavaScript 贴士和技巧，既有表单验证和 JSON 处理这样的基本技术，也有混搭和几何定位这样的应用实例。如果你想使用 JavaScript 编写更优秀的 Web 应用，我强烈建议你阅读本书。”

——Dylan Schiemann

SitePen 公司 CEO，Dojo 工具箱联合创始人

“现在市面上充斥着大量关于 JavaScript 的书，不过这些书大多都面向那些刚入门或是缺乏经验的 JavaScript 程序员。而本书并非如此，这本实用手册可谓出类拔萃。如果你是一个初学者，先去看些基础的书，等你具备一定经验后再读本书，效果会更好；如果你已经编写过不少 JavaScript 代码，你会在这本书中发现一系列你听过或见过的技术，但是本书会让你真正用好这些技术。总之，本书值得一读。”

——Stuart Langridge

kryogenix.org，@sil

引　言

只要你在过去几年中对 JavaScript 有一丁点留意，你就会听过这个说法：JavaScript 将会是下一个重量级语言。这个曾经被用于半成品实现和滚动信息展示的语言，如今已经演进为一个世界级的、面向对象的动态语言，无论是在客户端还是在服务器端，都有超高速的实现。

一方面，JavaScript 的设计者通过 EcmaScript 5（ES5）标准为 JavaScript 注入了新鲜健康的血液；另一方面，各种强大的 JavaScript 引擎（比如 V8、JavaScriptCore、SpiderMonkey、Rhino 和 Carakan）以及不断涌现的标准和技术（在这方面，CommonJS[①]和 Node[②]处于领先地位），使 JavaScript 既适合在浏览器端工作，也适合独立在服务器端构建强大的架构。甚至连微软最新推出的 Internet Explorer 9 也大幅提高了 JavaScript 的运行速度。

JavaScript 不仅仅是一门强大的动态语言，而且是一个庞大的生态系统，这个系统由它的开发工具、基础设施、框架和工具箱构成。JavaScript 功能全面，易于编写，适用于各种编程任务，尤其是那些基于 Web 的应用和服务。

是时候进入 JavaScript 的世界了！

本书内容和读者对象

本书并没有打算从语言的角度来讲解 JavaScript。首先，JavaScript 的语法并不复杂，只要你拥有一些主流编程语言的经验——哪怕只了解基本的概念（诸如变量、循环等），就已经足够了。本书并不需要你拥有 JavaScript 的经验（当然有的话会更好），更不要求你是编程专家。

事实上，如果你打算研究 JavaScript 的语言精髓或是核心技术细节，那最好还是去读一些专门的材料，Opera 提供的 Web 标准课程中的“JavaScript 核心技能”部分就很不错[③]。如果你需要

---

① 参见 http://commonjs.org/。
② 参见 http://nodejs.org/。
③ 参见 http://www.opera.com/company/education/curriculum/。

了解更复杂的语言实现细节，则可以去参考官方的语言标准，或是阅读那些圣经级的大部头，比如 David Flanagan 的 *JavaScript：The Definitive Guide*[①]。

针对常见的客户端 JavaScript 任务，本书提供了快捷且定性的解决方案，这些任务从简单（比如获取一个 DOM 元素的引用）到复杂（比如基于 Ajax 的自动完成）。这也意味着我们需要使用 JavaScript、CSS、DOM、Ajax、JSON 等技术。本书绝大部分在讲客户端（浏览器端）技术，服务器端的内容则很少。为了便于说明，本书也用了一些 PHP 脚本，不过你可以使用你喜欢的技术，比如通过 Node 用 JavaScript 编写服务器端程序！

不要直接照搬本书中的代码。本书每个任务都围绕着某个核心概念、潜在问题或技术技巧展开论述。你要理解这些概念、问题和技巧并灵活运用，而不要拘泥于特定的任务。最终，学完本书，你将成为一名更出色的 JavaScript 程序员。

本书与JavaScript库

坦率地说，如果你正在进行重要的 JavaScript 开发，却没有使用已有的优秀框架，那你就已经做错了。在浏览器端编写高效的网页脚本是一项挑战。你会碰到各种各样的障碍：DOM 中的不一致、错误的语言实现、CSS quirk 和诡异的 Ajax bug 等。在服务器端，即使已经有了运行时的支持，你仍需要把数据存储、网络栈和模块系统等基本服务器模块组合在一起，这涉及大量的工作。

幸运的是，优秀的开发者已经为你解决了这些难题。现在有大量的优秀框架（尤其在客户端）供你选择。附录 C 详细介绍了主流的 JavaScript 框架。

任何称职且注重实效的 JavaScript 开发者都会依赖一个或多个优秀框架，因此，对于那些在我看来相当“基本”的任务，本书会介绍它们在所有主流客户端框架中的实现方法。我选择的框架包括 Prototype、jQuery、MooTools、YUI、Dojo 和 ExtJS，它们覆盖了开发者心仪的大部分框架。

对于不那么“基本”的任务，我主要用我的最爱 Prototype[②]来处理（光箱特效除外，因为我认为在这个任务中 jQuery 插件更合适）。不过说实话，当你掌握了基本技巧之后，就可以用你常用的库来改写或重写我的解决方案。事实上，为了处理这种情况，我们在 GitHub 上建立了一个公用代码库[③]。因此，如果你需要用其他库（比如 jQuery），只需要简单地点一下 GitHub 的分支（Fork）按钮，而且查找这些代码库的派生版本也很容易。

---

① 此书中文版《JavaScript 权威指南》已由机械工业出版社出版。*

② 本书作者是 Prototype 的铁杆粉丝。*

③ 参见 http://github.com/tdd/pragmatic-javascript。

此外，本书所有代码都被打包并存档在本书网站上[①]。如果想测试这些代码的效果，请访问http://demo.pocketjavascript.com。

本书内容简介

本书按主题划分成不同的部分，每个部分均包含了一系列任务。书的末尾有一些附录，你在阅读正文之前很可能需要查阅其中一些内容，尤其是附录 A（JavaScript 快速参考）和附录 B（介绍 JavaScript 调试技术）。

- 第一部分包含了 JavaScript 开发者经常忽视但又非常重要的 JavaScript 编码模式。这些编码模式和框架关系不大，但为了编写良好的 JavaScript 代码，它们不可或缺。务必从这一部分开始阅读。
- 第二部分主要讲述前面提到的“基本”任务，主要包括基本的 DOM 和 CSS 操纵，以及事件处理和定时器的使用。由于它们都很“基本”，所以我列出了所有主流框架下这些任务的实现代码，你可以根据需要来选择。最好结合附录 C 阅读这一部分，从而对这些主流框架有一个全面的认识，作出可靠的决策。
- 第三部分主要讲用户界面，尤其是视觉效果和简洁的 UI 理念：漂亮的 tooltip、光箱特效、图片预载入、无限滚动[②]等。
- 第四部分是对第三部分的补充，主要讲绝大多数 Web 应用的重要组成部分——表单。此外，这部分会提到一系列用来辅助、简化和验证输入的工具。
- 第五部分主要讲客户端和服务端之间的关系，涉及的话题包括 cookie、JSON 以及 Ajax（域内和域间）。
- 第六部分是本书的最后一部分。这部分内容主要讲解如何利用混搭（mashup）思想和使用第三方服务。在这里，我选择了 3 个流行主题：Twitter 应用、Flickr 应用以及地理位置相关的 API。
- 附录 A 是我编写的一份 JavaScript 快速参考。其中总结了一些我认为很重要的 JavaScript 语言元素，并收录了一些实用的小贴士。希望它能对你有用。
- 附录 B 主要讨论如何调试 JavaScript。阅读这部分的所有内容是很有必要的，但愿它可以省去你在电脑前抓耳挠腮的大量时间，尤其是你使用 Internet Explorer 的时候。
- 附录 C 简要地介绍了本书收录的主流 JavaScript 框架。我已尽我所能地为这些框架逐一给出了精确的描述，并且展示出它们的特点，同时我还为如何选择 JavaScript 框架提供了一些建议。

---

① http://pragprog.com/titles/pg_js。

② 随着用户向下滚动页面，分批加载并显示更多内容。——编者注

- 附录 D 提供了一份快速参考，列出了有关 JavaScript 及其主流框架的帮助资源。这部分总结了附录 C 中的相关内容，并加入了一些额外资源（多数和 JavaScript 语言有关）。我把这部分放在本书的最后，以便查阅。

如何阅读本书

在本书中，每一个任务都编排成对开的两页：一页是文字解说，另一页则是代码。如果你阅读的是纸质版，这样的排版方式会让你感到流畅自然；如果你阅读的是电子版，请把阅读器设置成双页模式，以获得最佳的阅读效果。

致　　谢

写书从来不是一件容易的事情。尽管技术书不需要复杂的情节，也不需要绞尽脑汁地来充实每页内容，但是迫于同行的压力和职业的责任，技术书的作者需要尽可能保证内容的准确和实用，从而为读者提供最佳的实践。因此，编写技术书必须付出巨大的努力，而技术书作者也会竭尽所能来保证书的质量。

在编写本书的过程中，我首先要感谢本书中提及并用到的框架的创造者们：一方面，他们的作品使得 Web 前端工作变得更加轻松；另一方面，他们对本书的严格审阅保证了本书的质量。在这里，我特别感谢 Sam Stephenson、Thomas Fuchs、John Resig、Alex Russell、Jack Slocum 等核心框架开发者。此外，我还要感谢 Prototype Core 团队的成员。他们不仅乐于助人，更是非常杰出的开发者，我从他们身上学到了很多，这里尤其要感谢 Andrew Dupont、Tobie Langel 和 Juriy Zaytsev。

为了提高本书的技术准确性和整体质量，很多人（一部分刚才已经提到了）仔细审阅过本书。感谢他们的杰出工作，使本书的质量得到保证！我特别要感谢的是 Dion Almaer、Arnaud Berthomier、Aaron Gustafson、Christian Heilmann、Dylan Schiemann 和 Sam Stephenson。

这是我与 Pragmatic Programmers 合作出版的第二本书[①]。同以往一样，Dave Thomas 和 Andy Hunt 给了我这个宝贵的机会，使得我可以与 Pragmatic Guides 系列的优秀工作人员一起工作，他们是：Pragmatic Guides 系列编辑 Susannah Davidson Pfalzer、本书编辑 David McClintock（这是他在 Pragmatic Programmers 中的首次亮相）、眼光锐利的文字编辑 Kim Wimpsett、神奇的 Sara Lynn Eastler（她魔术般地完成了本书的索引）以及技术精湛的 Steve Peter（他精心排版了本书）。

最后，感谢我亲爱的妻子 Élodie。感谢她容忍我在过去五年中编写了四本书，并一直给予我支持和鼓励。她的爱让我感到自己是这个世界上最幸运的人，我无法想象没有她的生活将会是什么样子。所以，同以前一样，这本书为她而写。

---

① 本书作者的第一本 Pragmatic Programmers 书的书名为 *Prototype and script.aculo.us: You Never Knew JavaScript Could Do This!*，该书的中文译本《Prototype 与 script.aculo.us 终极揭秘》已由电子工业出版社出版。——译者注

目　　录

第一部分　JavaScript 必备操作

第二部分　DOM、事件及定时器

第三部分　UI 技巧

第四部分　表单技巧

第五部分　服务器端技术

第六部分　使用混搭

第七部分　附录

Part 1

第一部分

JavaScript 必备操作

序幕正式拉开。这部分介绍了 JavaScript 语言的必备操作，以便你在接下来的任务中大展拳脚。由于这部分中使用的都是纯粹的 JavaScript 语言，所以此处 3 个任务的代码不会依赖于任何框架或库。

- 通过任务 1 会学到如何动态访问对象的属性和方法（在代码里决定它们的名称）。
- 在任务 2 中，学习如何将代码控制在一个私有的作用域，从而避免“污染”别人的代码，同时也使得自己的代码具有自包含性。
- 最后，在任务 3 中，学习如何创建具有多样参数的函数。

别忘了，你可以在本书的网站上得到书中所有的源代码。你也可以直接在本书站点的演示页[①]中直接访问它们。此外，请准备好一个简单的空页面（在接下来的章节中可能需要加载一些库或框架），打开 JavaScript 控制台，以便测试书中的代码。

① 参见 http://demos.pocketjavascript.com。

任务 1 动态选择方法及属性

在实际工作中，我们经常会遇到这种情况：根据某个条件来调用两个方法[①]中的一个，或是在两个属性[②]中的一个上面进行读写操作。下面的代码展示了这种情形：

```
if (condition) {
  myObj.method1(someArg);
} else {
  myObj.method2(someArg);
}
```

JavaScript提供了一种简单的语法，即使用方括号操作符（[]）来动态地选择方法和属性。正如下面的代码所示，JavaScript有两种等价的成员访问语法（这个特征在动态语言里很常见）：

```
obj[expressionResultingInMembername] == obj.memberName
```

如果你曾用整数下标来访问数组中的某个元素，那你已经开始用方括号操作符来进行动态成员选择了。这是因为，数组对象本身就包含以数字下标命名的属性（以及`length`属性）。不过，JavaScript并不允许你使用点操作符（.）直接访问这些属性，因此`myArray.0`在语法上是非法的（太遗憾了，这本来是个挺酷的语法）。

为什么方括号操作符比点操作符表示法更强大呢？这是因为你可以在方括号中使用任何代表成员名称的内容来访问对象的成员。这些内容包括字面量、保存着成员名称的变量、名称组合（多数情况下是字符串的拼接）以及用三元操作符（`condition ? valueIfTrue : valueIfFalse`）实现的快速**`if/then`**选择。所有的这些内容都会被处理成一个字符串，然后JavaScript会用这个字符串来寻找对应的成员。

由于JavaScript中的函数本身也是对象，所以它可以像其他值一样被引用。如果一个表达式的结果是函数，你可以直接用括号操作符调用它，就像你直接用函数名称调用函数一样。

需要注意的是，如果你在向方法传递的参数上大量使用此类技巧，混乱的括号有可能会使代码变得难以阅读，此时使用常规的**`if/else`**结构更加明智。

---

① 方法是与对象相关联的函数。

② 属性是与对象相关联的变量。

使用方括号操作符

```
object['propertyName']      // => object.propertyName
object['methodName'](arg1) // => object.methodName(arg1)
```

切换行为

```
// 根据shouldBeVisible的值来调用show()或是hide()
element[shouldBeVisible ? 'show' : 'hide']();

// 避免IE中动画带来的巨大开销
// (假设这里已经有一个isIE变量)
element[isIE ? 'simpleEffect' : 'complexEffect']();
```

拼接方法名称

```
element[(enable ? 'add' : 'remove') + 'ClassName']('enabled');
```

请动手尝试下面代码

```
var love = { firstName: 'Élodie', lastName: 'Porteneuve' };
var useFirstName = true;
alert(love[useFirstName ? 'firstName' : 'lastName']); // => "Élodie"
```

任务 2　通过模块模式实现代码访问控制

随着你的JavaScript代码库的扩大,本应被控制在私有作用域中的函数和变量将会被暴露得越来越多，这时你的全局作用域被“污染”的可能性就会越来越大。这不仅会导致命名冲突（此处的脚本在无意中覆盖了其他脚本中的标识符），也会为bug提供了温床。

因此，我们需要编写自包含的、不透明的JavaScript代码，它不会向外界暴露内部的细节，也不会与现有的框架和库出现冲突。事实上，这正是“大规模编程”的主要要求。模块模式正是由此应运而生的。

模块模式的主要思想，是为那些通过**var**关键字声明的标识符和函数创建一个私有作用域，只有定义在这个作用域里的函数才能直接访问这些数据。为了使外界能够访问到函数里的部分内容，我们有两个选择。其一是返回一个包含选定值的对象（具体的做法见右页中的代码），然后把这个对象赋给外界的变量；另一种则是给函数传入一个外部作用域可访问的对象作为参数，使该函数能在这个对象中写入自己的属性（如果想让它的属性成为全局属性，只需传入`window`对象）。

JavaScript中，使用**var**关键字声明的标识符是*局部的*（它们只属于当前定义的作用域）。而未使用**var**关键字声明的标识符是*全局的*（它们会被嫁接到当前的默认作用域，而默认作用域在多数情况下就是全局的`window`对象）。

在一些特殊情况下，当前的默认域可能不是全局的，在这样的上下文中，即使使用非**var**声明的标识符，也不会对全局命名空间造成影响。不过，这已经超出了本任务所讨论的范围。

技术上来说，你并不需要像命名私有标识符那样显示命名“公有属性”。事实上，你完全可以在返回的对象中用匿名函数的方式定义公有方法。但这样做的话会使得代码晦涩难懂（或是带有误导性），更重要的是，这会使调试难以进行。因此，尽可能用*命名函数表达式*来定义你的函数：

```
function myFunctionName(...) { ... }
```

这会令代码更加清晰，同时也便于在调试时观察栈轨迹。

在匿名函数中使用var关键字

```
(function () {
  var privateField = 42;

  function innerFunc() {
    notSoPrivate = 43;
    return notSoPrivate;
  }

  alert(privateField); // => 42
  innerFunc();
  alert(notSoPrivate); // => 43
})();
alert(typeof privateField); // => Undefined
alert(notSoPrivate); // => 43 (变量被泄露到外部)
```

试试这个例子："私有属性"

```
var obj = (function() {
  var privateField = 42;
  var publicField = 'foobar';

  function processInternals() {  alert('Internal stuff : ' + privateField);}

  function run() {
    processInternals();
    alert('Still private stuff: ' + privateField);
    alert('Public stuff: ' + publicField);
  }

  return {
    publicField: publicField,
    run : run
  };
})();

obj.run() // 弹出三个对话框:Internal, still private, public
obj.publicField // foobar
obj.processInternals() // Undefined
obj.privateField // Undefined
```

任务3 使用可选/可变/命名参数

为了熟练掌握JavaScript中参数的使用技巧，你首先需要意识到：在JavaScript中，你所显式声明的形参并不会对实参造成限制。因为每个函数都会把它的实参保存在一个预定义的arguments变量中（arguments具有length属性和[]操作符[①]）。因此，形参实际上只是为实参提供了本地名称。如果形参与实参的数量一致，那这些形参将会引用实参的内容。如果不是的话，空缺的形参会被赋予undefined。

现在，请注意下一页中可选参数的例子。我用undefined === rant测试第二个实参是否存在。为什么要连用三个等号呢？这就得说说JavaScript混乱的等价规则了。请看下面的表达式：

```
undefined === null // => false
undefined == null  // => true
```

看明白了吧。因此，假设我们认为**null**是rant的一个合法值，我们就需要同时检查rant的值和类型，这正是 === 比较操作符所做的，它是严格比较操作符（strict equality operator）。

在多数情况下，你需要为参数的“空值”给出更明确的定义。比如，假设我们需要rant是一段有意义的文字：空字符串、**null**、**undefined**、**0**和**false**显然都是没有意义的。正巧这些值在JavaScript中都与false等价，因此，我们可以用相当简洁的方式来设置rant的默认值：

```
rant = rant || 'IE6 must die!';
```

鉴于JavaScript的false等价关系是如此混乱，因此我在右页的第四个例子中用了**in**操作符来判断options对象是否包含一个给定的属性，而不是用 !options[opt]。因为这段代码相当通用，很多地方都会用到它，所以我采取了一种相对保守的编码风格：在使用某个属性之前，先判断这个属性是否存在。

这个例子同时也展示了如何用**for...in**语法遍历一个对象的属性（property）。

最后，请注意，我是通过该函数的公有属性来设置repeat()的默认参数的，这样，用户就可以在不触动全局对象的情况下随时修改该函数的默认参数。为了获得函数自身的引用，我们会使用arguments的特殊属性callee[②]。

---

① 尽管很像数组，但实际上它并非数组。*

② arguments的callee属性会返回正被执行的Function对象。*

声明参数（命名参数）

```
function repeat(rant, times) {
  while (--times >=  0)
    alert(rant);
}
repeat('IE6 must die!', 5); // => 连续弹出5个对话框
```

动态获得不定数量的参数

内置的`arguments`变量允许你动态访问函数的参数。

这使你可以模拟其他语言中的变长参数列表，比如C语言的`varargs`。

```
function repeat(times) {
  while (--times >= 0) {
    for (var index = 1, len = arguments.length; index < len; ++index) {
      alert(arguments[index]);
    }
  }
}
repeat(2, 'IE6 must die!', 'So should IE7...'); // => 连续弹出4个对话框
```

为可选参数设置默认值

```
function repeat(times, rant) {
  if (undefined === rant) {
    rant = 'IE6 must die!';
  }
  while(--times >= 0) {
    alert(rant);
  }
}
repeat(3); // => 连续弹出3个有关IE6的对话框
repeat(3, 'So should IE7...'); // => 连续弹出3个有关IE7的对话框
```

用字面量对象实现伪命名参数

```
function repeat(options) {
  options = options || {};
  for (var opt in (repeat.defaultOptions || {})) {
    if (!(opt in options)) {
      options[opt] = repeat.defaultOptions[opt];
    }
  }
  for (var index = 0; index < options.times; ++index) {
    alert(options.rant);
  }
}
repeat.defaultOptions = { times: 2, rant : 'IE6 must die!' };

repeat(); // 弹出两个与IE6有关的对话框
repeat({ times: 3 }); // 弹出3个与IE6有关的对话框
repeat({ times: 2, rant: 'Flash must die!' }); // 弹出两个与Flash有关的对话框
```

Part 2 第二部分

DOM、事件及定时器

通过对第一部分的学习，我们对 JavaScript 已有了一定的认识，接下来，让我们开始探索 JavaScript 和 Web 页面之间的纽带：DOM 操纵。

DOM 操纵一般分为以下几类。

- 获得 DOM 元素[①]的引用，以便进行操作。这部分内容将在任务 4 中阐述。
- 改变 DOM 元素的外观，可以是直接修改，也可以是使用动画（大多会通过显示、隐藏或移动元素来修改样式）。任务 5 将会讲述这部分内容。
- 修改 DOM 元素的内容。任务 6 讲述了这方面的内容。

由于上面的操作都是在页面初始化时、响应特定的事件或在一段时间发生后得以执行，因此我们还需要讨论以下内容。

- 任务 7 阐述了页面初始化的概念。具体地说，我们利用这些概念来检测 DOM 是否载入完成，以便对其进行操作。
- 任务 8 提供了事件监听的基本概念，任务 9 提供了高效监听事件的方案，而任务 10 则进一步阐述了如何发挥事件的威力。
- 任务 11 展示了如何使用定时器（给出了一个通过定时器模拟后台处理的例子）。

由于上面的操作都是 Web 应用中所不可或缺的，所以我在这里提供了这些操作在所有主流 JavaScript 框架[②]下的代码。同样的操作在不同框架下的实现可能略有差异，但是它们在功能上是等价的。对比剖析[③]在几个世纪前是很流行的学习方法，现在它也不失为一种研究问题的途径，能够使你获得更宽广的视野。

---

① 本书统一将 element 译为元素，将 node 译为节点。*

② 也就是本书作者所选择的框架，包括 Prototype、jQuery、MooTools、YUI、Dojo 和 ExtJS。*

③ 原文为 compared anatomy，这是一种生物学的研究方法，请访问 http://en.wikipedia.org/wiki/ Comparative_anatomy 以获得更多细节。*

任务 4 获得 DOM 元素的引用

获得页面上的DOM元素以及“遍历”DOM（从一个元素移至另一个元素）是编写Web脚本的必需操作。虽然现在的浏览器已经可以提供不错的DOM元素获取方法（尽管人们用了足足十年才达到这一步），不过在不使用库的情况下遍历DOM仍然非常烦琐。因此在实际工作中，我们都需要使用这样或那样的库。

在这里需要注意以下几点。

首先，要注意`document.getElementById`，任何依赖于这个方法的代码都会成为IE怪异行为的牺牲品。因为在IE中，这个方法也会通过`name`属性来寻找匹配的元素。如果你使用的是严格的`DOCTYPE`（你应该这么做），这看起来意味着只需要“注意字段的命名”。由于你已经在潜意识中排除了`<head>`和`<meta>`标签的`name`属性，所以当你不知不觉地定义了一个`id`属性和`<meta>`标签特定属性相同的元素后，如果你用`document.getElementById`获取它，你将会惊讶地发现你得到的是`<meta>`元素！所以，尽可能避免使用这些特殊值作为`id`属性的值。

在右页，我提到了绝大多数选择元素的技术，都需要先指定*上下文节点*或*根节点*，然后以这个节点作为根元素，在其之上进行选择操作。默认的上下文是整个文档。翻阅一下API文档，你会发现诸如`$$()`、`query()`以及`$()`等方法都有一个可选参数（一般是第二个），以便你来指定上下文元素。要记住，搜索范围越窄，选择的速度也就越快。

最后，原始的DOM API并不适合实际的元素访问操作，因为它们适用于*节点*，而不是*元素*。原始的DOM API会让你陷入到空白节点、注释节点、文本节点等节点的泥潭之中，因此，绝大多数库提供了不错的面向元素的辅助方法。Prototype、jQuery和MooTools都提供了丰富的API，这包括`previous()`（或`prev()`）、`next()`、`siblings()`、`ancestors()`等方法，还包括那些以`get`为前缀的同类方法。YUI 3 API中提供的方法略微少些，而Dojo和Ext JS则暂时缺乏此类辅助方法。

绝大多数的库都支持基于ID和基于CSS3的选择。

通过ID属性获得对应的元素

```
document.getElementById('elementId')          // 原始的W3C DOM
$('elementId')                                // Prototype， MooTools
$('#elementId')                               // jQuery
Y.one('#elementId')                           // YUI3
dojo.byId('elementId')                        // Dojo
Ext.getDom('elementId')                       // Ext JS3
```

通过XPath/CSS选择来获得元素

不同的库所支持的语法可能会有所差异，W3C选择器API只在最新的浏览器中可用（不过其速度相当快），这些浏览器包括：Firefox 3.1+、Safari 3.1+、IE8+（标准模式）、Chrome以及Opera 10+。

还需要注意的是，这些库都提供了一些途径来指定上下文，也就是作为搜索起点的根节点(默认是整个文档)。尽可能缩小上下文的范围，以争取更快的执行速度和更小的内存占用。

```
document.querySelectorAll('selectors')        // Native（如上）
$$('selectors')                               // Prototype， MooTools
someRootElement.select('selectors')           // Prototype
$('selectors')                                // jQuery
Y.all('selectors')                            // YUI 3
dojo.query('selectors')                       // Dojo
Ext.query('selectors')                        // Ext JS 3
```

元素间移动（遍历DOM）

下面的例子基于Prototype，如果需要这个例子的上下文和额外的信息，请阅读左页。

```
// 从具有指定id属性的元素开始，到达最近的class属性为category的<h2>元素
// 然后遍历这个元素的兄弟元素直到遇到一个class属性为summary的元素
$('someDeeplyNestedElement').up('h2.category').next('.summary');

// 找到带有名为sifr的CSS class的元素
// 设置这些元素的直接容器的CSS文本缩进属性
$$('a.sifr').invoke('up').invoke('setStyle', 'text-indent: -9999px');
```

任务 5 动态修饰内容

获得某个DOM元素的CSS属性的当前值（无论是指定的还是计算得出的），并修改这些值以改变页面的外观，是创建Web UI所不可或缺的操作。不过，浏览器和W3C的标准在这方面上还是一片空白，因此这个重任就落到了库和框架的肩上。在右页你会发现，现在的库均为这些操作提供了丰富的API，而且它们彼此十分相似。

如何进行动态修饰呢？答案就是使用`class`属性。

为了开发属于你自己的视觉效果，你当然可以自己动手修改样式，但这个工作最好还是留给那些库的作者和JavaScript大师。我相信，作为一个前端开发者，你能够把特定的样式和JavaScript代码清楚地分离开来，然后通过切换CSS的`class`名称和使用你所编写的特效，来获得你想要的效果。

事实上，所有的库都提供了一系列统一的API来增加、删除和检查某个元素上的CSS的`class`名是否存在。MooTools、jQuery、YUI、Dojo和Ext都可以通过`addClass()`、`hasClass()`、`removeClass()`及`toggleClass()`来完成这些操作。Prototype则只是在这些方法的名称后面加了一个`Name`后缀（比如`hasClassName()`）。

不要忘了使用快捷操作！

所有的库都为常用的操作提供了一系列快捷的读/写操作，比如操作可见性（隐藏/显示/切换）、透明度（依照CSS标准，即使在IE上也是这样）及维度等。阅读文档时，你就会发现：一般而言，方法的名称就能说明它的含义。

设置元素的样式

```
// Prototype
$(element).setStyle('prop: value; prop2: value2;')
$(element).setStyle({ prop: 'value', prop2: 'value2' })
// jQuery
$(element).css('prop', 'value')
$(element).css({ prop: 'value', prop2: 'value2' })
// MooTools
$(element).setStyle('prop', 'value')
$(element).setStyles({ prop: 'value', prop2: 'value2' })
// YUI 3
Y.one('#id').setStyle('prop', 'value')
Y.one('#id').setStyles({ prop: 'value', prop2: 'value2' })
// Dojo
dojo.style(element, 'prop', 'value')
dojo.style(element, { prop: 'value', prop2: 'value2' })
// Ext JS
Ext.get(element).setStyle('prop', 'value')
Ext.get(element).setStyle({ prop: 'value', prop2: 'value2' })
Ext.get(element).applyStyles(function(e) { return someSpec; })
```

获取元素的样式

```
// Prototype
$(element).getStyle('prop')
// jQuery
$(element).css('prop')
// MooTools
$(element).getStyle('prop')
// YUI 3
Y.one('#id').getComputedStyle('prop')
Y.one('#id').getStyle('prop')
// Dojo
dojo.style(element, 'prop')
dojo.style(element) // => 获得该元素所有的样式
// Ext JS
Ext.get(element).getStyle('prop')
Ext.get(element).getStyles('prop', 'prop2', 'prop3')
```

相关任务

- 任务6

任务 6 修改元素的内容

你可能会问，为什么不通过设置innerHTML属性来修改该元素的内容呢？

直接把指定的超文本赋给容器元素的innerHTML属性，这样看起来很有吸引力。确实，这样做会迅速获得几个数量级的速度提升。但是，在许多浏览器中，一些不规范的超文本可能会导致innerHTML属性失效，而库则会在需要的情况下使用其他机制（比如手动DOM构建或超文本预处理）来保证超文本能顺利插入。当使用内容注入而非全部的内容替换，或者是包含内联脚本的超文本时（接下来的段落中将会讨论这一内容），使用库更加方便、简洁，并且不容易出错（相对与手动操作超文本而言）。

注意，不要混淆*替换*（updating）和*更新*（replacing）这两个操作。大多数方法默认执行的是更新操作——举例来说，dojo.place()方法中的only位置。更新操作会改变元素内部的内容，而替换操作则会改变*元素自身*，因此可能会导致元素的ID和与之关联的事件监听器失效。替换操作实际上和设置outerHTML属性相等价。

幸运的是，一些库提供了一系列特殊方法来辅助基本的更新/插入操作，比如元素包装、基于选择器的多元素操作以及清除多余的空文本节点等。因此，请务必阅读你常用的库的API文档，以了解更多细节。

不过，如果要注入<script>标签，那你可得小心了。因为根据默认设置，本地注入的超文本中内嵌的<script>标签中的脚本并不会运行，有时这可能会导致一些问题，所以如果可能的话，尽量利用事件代理[1]实现等同的功能。不过，由于这个需求是如此普遍，一些库已经提供了相应的机制。

- Prototype的update()、replace()和insert()方法默认会运行内联的<script>标签中的脚本（为了Ajax更新，需将evalScripts选项设为**true**）。
- jQuery的html()方法默认也会执行内联的脚本。
- Ext的update()方法，把第二个参数设置为**true**，以运行内联的脚本。

出于安全方面的考虑，这些待运行的脚本有时可能会在一个特殊的执行上下文中运行。请阅读你常用的库的文档，以获得更多的细节。

---

① 阅读任务9以获得更多细节。

更新元素的全部内容

```
// Prototype
$(element).update('<p>new internal HTML</p>')
$(element).replace('<p>This will replace the element itself</p>')
// jQuery
$(element).html('<p>new internal HTML</p>')
$(element).text('The <div> and <span> elements carry no inherent semantics.')
// MooTools
$(element).set('html', '<p>new internal HTML</p>')
$(element).set('text', 'The semantics of << and >> varies across languages.')
// YUI 3
Y.one('#id').setContent('<p>new internal HTML</p>')
// Dojo
dojo.place('<p>new internal HTML</p>', element, 'only')
dojo.place('<p>This will replace the element itself</p>', element, 'replace')
// Ext JS
Ext.get(element).update('<p>new internal HTML</p>')
```

请注意，jQuery和MooTools使用了特殊的更新机制，来对传入文本中的标签进行转义，这使得在页面展示代码变得很方便。

向元素中注入其他内容

```
// Prototype。位置包括：'before', 'top', 'bottom', 'after'
$(element).insert('<p>This gets at bottom</p>')
$(element).insert({ pos: markup, pos2: markup2... })
// jQuery（提供了大量的方法来控制插入的位置）
$(element).before('<p>This gets before the element</p>')
$(element).prepend('<p>This gets at top</p>')
$(element).append('<p>This gets at bottom</p>')
$(element).after('<p>This gets after the element</p>')
// YUI 3
Y.one('#id').prepend('<p>This gets at top</p>')
Y.one('#id').append('<p>This gets at top</p>')
Y.one('#id').insert('<p>This gets where told</p>', nextChildElement)
Y.one('#id').insert('<p>This gets where told</p>', childIndex)
// Dojo。位置包括：'before', 'first', 'last', 'after'
dojo.place('<p>This gets where told</p>', element, pos)
// Ext JS。位置包括：'beforeBegin', 'afterBegin', 'beforeEnd', 'afterEnd'
Ext.get(element).insertHtml(pos, '<p>This gets where told</p>')
```

相关任务

- 任务5。

任务 7　在 DOM 加载完成后运行脚本

为了使你的页面在一开始就能给出正确的响应，你的脚本应该在页面DOM加载完成后尽快得到执行。

如果你使用`window`的`load`事件，你的脚本就得等到所有资源（包括样式表、图片以及Google Analytics[①]跟踪器之类的加载时间很长的脚本）加载完成之后才能被执行。而这往往会使页面得不到及时的渲染。

所有的JavaScript库都在这方面下足了功夫，它们的处理方法很相似：以自定义方法或是自定义事件（比如Prototype、MooTools和YUI3）的形式提供了一个触发器，你只需把要执行的脚本写到一个函数里，再把这个函数指派到这个触发器上就可以了。

顺便提一个良好实践：如果你的UI部分依赖于JavaScript，那你就应该把非JavaScript相关的UI设置为默认可见，而把JavaScript相关的UI设置为隐藏。等DOM加载的事件发生之后，你只需把JavaScript相关的部分添加到文档中，然后利用CSS切换其可见性。由于这样可以避免页面载入中的UI闪烁，因此它比直接使用JavaScript来隐藏无关UI要好很多。

现在举一个相当有技巧的例子：假设你的初始化代码位于某处的实例方法中，为确保实例方法内部的`this`指向正确的实例，需要保持方法与原来实例的绑定关系。多数库提供了常规方法绑定的方式（比如Prototype中的`bind()`）来实现它，而Dojo和Ext JS在这方面走得更远，它们允许你为方法的实例提供一个显式的引用。也就是说，它们可以直接在合适的上下文中调用方法：

```
// Dojo
dojo.addOnLoad(binding, fx)
// Ext JS
Ext.onReady(fx, binding)
```

你也应该去检查你的初始化代码是否真的在DOM加载时执行；在少数情况下，一些代码会过早执行。举例来说，你可能需要等待某幅图像载入完成，以便根据它们的尺寸来设置UI；与此类似，你可能需要等待某个CSS样式表加载完毕，以便设置元素的尺寸、颜色等属性。在这种情况下，你有可能要使用`window`的`load`事件。不过，在大多数情况下，DOM加载事件已经够用了。

---

① Google Analytics是Google为网站提供的数据统计服务。可以对目标网站进行访问数据统计和分析，并提供多种参数供网站拥有者使用。*

在DOM加载时执行指定脚本

```
// Prototype
document.observe('dom:ready', fx)
// jQuery
$(fx)
// MooTools
window.addEvent('domready', fx)
// YUI 3
Y.on('domready', fx)
// Dojo
dojo.addOnLoad(fx)
// Ext JS
Ext.onReady(fx)
```

相关任务

- 任务8

任务 8 监听及停止监听事件

事件是一个庞大的主题，短短两页的篇幅连基本概念都无法覆盖。因此，这里只提供一个非常简短的快速参考（位于右页）。你应该（我几乎敢断言，你必须）花时间，找到你使用的库的文档或指南，将其中与事件相关的内容仔细阅读几遍。不要觉得这是浪费时间，掌握事件的技巧会为你带来巨大的回报。

需要特别注意的是，多数库为监听特殊事件提供了简写方法，因此，你应该调用的是 `onclick(handlerFx)` 而不是 `connect('click', handlerFx)` 。另一个有用的技巧是，多数库允许用较少的参数来停止或关闭一系列事件（例如，关闭所有的单击事件或停止监听某个元素上的所有事件）。

值得注意的特例：Dojo使用一种单独的机制来连接任意形式的事件（如常规的DOM事件、自定义事件、所谓的全局事件、发布–订阅事件以及单纯的方法调用等）和任意形式的函数（包括真实的事件处理器、普通的函数和方法）。这是个非常好的特性。而Ext JS可以通过设置`on()`的第四个参数，来配置各种各样的事件。

所有的库都提供了一种简单的方式，以便你在文档级别来监听事件（利用事件冒泡）。举例来说，你可以使用Prototype中的`document.observe`，或者利用`dojo.doc`或`Ext.getDoc()`这些包装器。不过，你不应该过度依赖事件捕获（自上而下的事件传播/检查）相关的参数和选项，因为在IE8之前的IE中，它们没有得到很好的支持。

所有的库均为事件处理器提供了一个加强版的`Event`对象，其中既包含了W3C要求的属性和方法，也包含了一些辅助性的属性和方法。

默认情况下，事件处理函数在作为函数引用传递时会丢失它们潜在的绑定关系，而在全局上下文中执行[①]。为了解决这个问题，一些库允许你传递一个可选的参数作为作用域对象。

最后，绝大多数库支持自定义事件和一些惯用事件。就我个人而言，DOM提供的一些备用事件不可或缺，比如`mouseenter`和`mouseleave`等事件。

---

① 作者的这篇文章包含了很多JavaScript绑定的要点和技巧，网址在http://www.alistapart.com/articles/getoutbindingsituations/。

在某个元素上监听某个事件

```
// Prototype
$(element).observe('event', handlerFx)
// jQuery
$(elementOrSelector).bind('event', handlerFx)
// MooTools
$(element).addEvent('event', handlerFx)
// YUI 3
Y.on('event', handlerFx, elementOrSelector)
// Dojo (handlerFx既可以是上下文无关的函数，也可以是上下文相关的方法)
dojo.connect(element, 'event', handlerFx)
// Ext
Ext.get(element).on('event', handlerFx)
```

在多个元素上监听某个事件

```
// Prototype
elements.invoke('observe', 'event', handlerFx)
// jQuery
$(elements).bind('event', handlerFx)
// YUI 3
Y.on('event', handlerFx, elementOrSelector)
// Dojo
dojo.query(selector).connect('event', handlerFx)
// Ext, for on-DOM-readiness bindings:
Ext.addBehaviors({ 'selector@event': handlerFx... })
```

停止监听

```
// Prototype
$(element).stopObserving('event', handlerFx)
// jQuery
$(elementOrSelector).unbind('event', handlerFx)
// MooTools
$(element).removeEvent('event', handlerFx)
// YUI 3
Y.detach('event', handlerFx, elementOrSelector)
// Dojo
dojo.disconnect(handleReturnedByConnect)
// Ext (注意，这里是"un"而不是"on"……)
Ext.get(element).un('event', handlerFx)
```

相关任务

- ❑ 任务7
- ❑ 任务9
- ❑ 任务10

任务9 利用事件委托

请用心理解并牢记这句话：优先使用事件委托。

你应该知道，绝大多数的事件可以冒泡，比如说鼠标或键盘事件。当这些事件在DOM某处发生时，它们会沿着祖先元素这条线，从内向外依次触发各个元素，直到文档的根元素（如果这些元素中的任何一个都没有监听器来阻止冒泡继续的话）。

假设我们有大量的元素，而且这些元素需要共享同一个行为。如果需要在这些元素上监听事件，我们最好在DOM的高层次，也就是离这些元素最近的公共祖先上或在`document`上监听事件。这样可以节省大量的内存和CPU时间。

在DOM的高层监听事件对加载Ajax内容的行为也有好处。由于你在加载内容的“外面”进行监听，因此新添加进来的元素会自动获得现有的行为，而不需要在加载后设置额外的监听器。

不过，有时我们的触发事件不会冒泡，这时我们就需要对事件委托进行一些hack，或是完全抛弃事件委托。毫无疑问，`submit`、`focus`和`blur`这些事件不会冒泡——这使得对共享表单及域的处理变成了一场噩梦。尽管存在一些代码来模拟这些事件的冒泡，但这些事件的不可冒泡性仍令人感到不爽。

需要指出的是，直到1.4版本，jQuery的`live()`方法才开始在本质上灵活地支持事件委托，而且不会带来性能问题。也是从那个版本开始，jQuery增加了操作非冒泡事件的能力。此外，Dojo的`behavior()`看起来像是使用了事件代理，但其实并非如此。实际上它只是一个不错的语法糖。

注意示例代码第2行上的`findElement()`调用。当前，我们的链接只包含一段简单的文本，所以在链接内部单击也就意味着单击链接本身。不过，假设你在这个链接上面加了一个图标（比如说一个加/减图标），那么单击可能发生在图标上，但它仍然是单击链接事件。因此，在Prototype中指派一个基于委托的事件代理处理器时，请使用事件的`findElement()`方法，而非不太智能的`element()`方法。

切换条目内容

dom/delegation.html

```
<ul id="items">
  <!—我们将使用JS在每个LI中插入切换器 -->
  <li><div><p>Data 1</p><p>Data 2</p></div></li>
  <li><div><p>Data 1</p><p>Data 2</p></div></li>
  <li><div><p>Data 1</p><p>Data 2</p></div></li>
  <!-- 下面可能会有更多的元素…… -->
</ul>
```

这是一段基于Prototype的脚本：

dom/delegation.js

```
$('items').observe('click', function(e) {
  var trigger = e.findElement('a.toggler');
  if (!trigger) return;
  e.stop();
  var content = trigger.up('p').next('div');
  if (!content) return;
  content.toggle();
  trigger.update(content.visible() ? 'Close' : 'Open');
  trigger.blur();
});

$('items').select('li').each(function(item) {
  item.insert({ top: '<p><a class="toggler" href="#">Open</a></p>' });
  item.down('div').hide();
});
```

使用事件代理

```
// Prototype 1.7
$('items').on('click', 'a.toggler', handlerFx);
// jQuery 1.4
$('a.toggler', '#items').live('click', handlerFx);
// YUI 3
Y.delegate('click', handlerFx, '#items', 'a.toggler');
// Ext
Ext.get('items').on('click', handlerFx, this, { delegate: 'a.toggler' });
```

相关任务

- 任务8

任务 10 将行为和自定义事件解耦

当你的代码库增长到一定的规模，或者你打算在一个完全不同的上下文复用部分代码时，你可能就会碰到这样的情形：不管DOM是什么样子，我希望这段代码在这样的情况下被触发。

多数情况下，这个问题涉及微件（widget）间的交互。比如说，某些后台聊天轮询引擎会通知聊天部件收到了新信息，或者当图片浏览窗中的某个图片被单击后，对应的图片浏览控件和预载入图片放大控件均会被告知这个操作的发生。

这就是自定义事件的概念。根据你使用的框架不同，自定义事件的行为在某些方面会有所差异。

- **类似DOM的行为**：你在DOM节点（包括`document`对象）监听并触发自定义事件。这些事件既可以冒泡，也可以被拦截。这正是Prototype、jQuery和MooTools所做的。如果事件不能扩散，就必须在触发事件的对象上进行监听。
- **命名空间**：一些框架需要你为你的事件指定命名空间，通常使用一个冒号前缀来把你的事件和原生事件区分开来。Prototype强制要求这种做法；其他框架则允许你按照你习惯的方式命名自定义事件。
- **自定义额外数据**[①]：JavaScript框架允许你在触发自定义事件时，向事件处理器传送额外的数据。Prototype接受一个单独的额外参数（当然，这个参数里可以包含各种各样的内容），以作为事件`memo`属性的内容。除了初始事件对象参数以外，jQuery可以向事件处理器传递任意数量的额外参数。其余的框架会把这些参数在不加前缀的情况下传递给事件处理器。Dojo需要把待传的参数放到一个数组里传递，即使参数只有一个也是如此。
- **通用事件API**：只有Dojo保留了操作原生DOM事件的正常API，而操作自定义事件则需要特殊的发布/订阅API。这也意味着Dojo中的自定义事件不具有DOM事件的一些行为（比如冒泡）。
- **声明**：我们往往需要在预定义的事件中加入一些特殊的变化（例如，需要Alt键按下才能触发的单击事件），MooTools允许你定义此类自定义事件。此类事件需要预先声明，即便你只是声明它们的名字。任何未声明的自定义事件都不会被触发。

① 原文为payload，在这里它指的是传给事件处理函数的额外数据。*

监听一个自定义事件

```
// Prototype --通过event.memo传入额外数据
$(element).observe('ns:event', handlerFx)
document.observe('ns:event', handlerFx)
// jQuery -- 通过多余的参数传入额外数据
$(elementOrSelector).bind('event', handlerFx)
// MooTools -- 通过事件处理器参数传入额外数据
Element.Events.event = {};
$(element).addEvent('event', handlerFx)
// YUI 3 -- 通过事件处理器参数传入额外数据
Y.on('event', handlerFx)
// Dojo -- 通过事件处理器参数传入额外数据
dojo.subscribe('event', context, handlerFx)
```

触发一个自定义事件

```
// Prototype
$(element).fire('custom:event');
document.fire('custom:event');
whichever.fire('custom:event', { foo: 'bar', baz: 42 });
// jQuery
$(elements).trigger('event')
$(elements).trigger('event', { foo: 'bar', baz: 42 });
$(elements).trigger('event', ['bar', 42]);
// MooTools
$(element).fireEvent('event')
$(element).fireEvent('event', arg)
document.fireEvent('event', [arg1, arg2, arg3])
// YUI 3
Y.fire('event')
Y.fire('event', arg1, arg2, arg3)
// Dojo
dojo.publish('event')
dojo.publish('event', [arg])
```

相关任务

- 任务8

任务 11 模拟后台处理

有时，你会需要在网页里进行一个相当耗时的处理，比如，从一个表中得到一个数据集，经过各种计算（可能需要相当长的时间），最后得到一个图表。你希望在这个过程中页面仍然能够得到正常的响应，这就需要用到后台处理技术。

这是因为JavaScript引擎会以如下方式执行你的代码。

- JavaScript本质上是单线程的。
- 你的JavaScript运行线程实际上和你的页面共享了同样的资源。这也意味着，当你的JavaScript代码运行的时候，任何页面渲染都不会发生。新的内容不会出现，内容无法重排，甚至被其他窗体所遮挡的页面也无法被重绘……总之，什么都不能做。

所以，如果你的页面在执行一些密集的计算，那么直到处理结束前页面都不会有响应。这通常意味着整个浏览器都会失去响应。为了防止出现这种情况，一些浏览器提供了“中断运行时间过长的脚本”的功能。而其他浏览器，例如Chrome，则会为每个页面开一个独立的进程，以处理这个问题。

如果不使用Web Workers[①]（至少现在来看，这显然不是一个跨浏览器的选择）的话，你就需要使用一些伪并行处理的技巧，此类技巧一般依赖于全局`window`对象提供的一对方法——`setTimeout()`和`clearTimeOut()` 。

这些技巧的思路是把一个大型任务分解成若干个小步骤，然后一边执行这些步骤，一边记录任务的进度，并在固定的时延对这些步骤进行调度。当一个步骤完成之后，经过一段时间再启动下一个步骤。在这段空闲的时间里，浏览器会恢复对页面的控制，因此就可以正常地处理页面行为，并运行其他待执行脚本。

虽然调用`clearTimeout()`来清理调用`setTimeout()`时所存储的定时器处理器并不是必需的，但这是一个良好的编码实践，它可以减少内存的消耗，而且不会带来多少性能上的开销。

尽管这个技巧适合用于密集计算处理任务，但是它并不适合那些需要平滑过渡的行为，比如视觉效果，因为定时器的精度很差（在25ms~500ms变化）。在这种情况下，你需要使用一个具有精确固定间隔的定时器[②]。

---

① 详见http://en.wikipedia.org/wiki/Web_worker。*

② 如果想要了解具体的实现方式，请阅读Thomas Fuch编写的这个50行的绝妙程序Émile：http://github.com/madrobby/emile。

调度及停止代码的执行

利用定时器模拟后台处理需要两个核心方法：

```
var handle = window.setTimeout(callback, intervalInMs);
window.clearTimeout(handle);
```

让用户切换后台处理

本书的在线源码库（dom/background.js）中包含了一个完整的演示。

下面是这个例子中的关键代码：

dom/background.js

```
var CHUNK_INTERVAL = 25 ; //单位为毫秒
var running = false, progress = 0, processTimer;

function runChunk(){
  window.clearTimeout(processTimer);
  processTimer = null;
  if (!running) return;
  // 模拟任务的各个步骤
  for (var i = 0; i < 10000; i += (Math.random() * 5).round())
    ;
  ++progress;
  updateUI(); //用来更新一个进度条，请到在线源码库了解更多的细节
  if(progress < 100){
    processTimer = window.setTimeout(runChunk, CHUNK_INTERVAL);
  } else {
    progress = 0, running = false;
  }
}

function toggleProcessing (){
  running = !running;
  if (running) {
    processTimer = window.setTimeout(runChunk, CHUNK_INTERVAL);
  }
}
```

Part 3 第三部分

UI 技巧

操作 DOM 是我们的基本工具，每个操作最终都是在操作 DOM。但实际的需求往往要比单纯操作 DOM 更抽象，比如实现有用的用户界面（UI）特效、功能和微件，有效地处理表单等。在这一部分，我们将开始学习这些实用的 UI 功能。

- 任务 12 会探讨如何制作既好看又与上下文相关的“泡泡”提示。
- 尽管弹窗的效果不错，但在不能用 JavaScript 的情况下也应该要能访问那些内容才对。在任务 13 中可以看到怎样同时处理好这两种情况（启用 JavaScript 和禁用 JavaScript）。
- 图片预载入并不总是能用 CSS spriting 实现（比如预载入用户提供的图片）。为了处理这种情况，任务 14 说明了如何用 JavaScript 做到这一点。
- 光箱（通过暗化页面的其余部分而被突出显示的伪对话框）的合理使用可以创造简洁的用户体验。任务 15 展示了最佳的解决方案。
- 有时，大数据集浏览用智能翻页实现效果更好。任务 16 阐明了这一点。
- 有时，你需要载入新的内容，而这些内容要加在触发载入的元素所在页面位置的上方。为了避免页面晃动，看看任务 17。

本书后面会用框架和库来减轻工作量。一般我都用 Prototype（但光箱特效的任务用 jQuery 插件，因为它看起来是完成这个特殊任务的最佳工具）。

诚然，Prototype 不一定吸引每位前端开发者，所以看看本书代码的 GitHub 库吧。[①]我的目的是让大家能创建对应的分支（forks），用自己最喜欢的 JavaScript 框架来改写。这样一来，你就能得到用你喜欢的框架写的代码。如果还没人写出你喜欢的框架，你不妨自己着手写一个。

① 参见 http://github.com/tdd/pragmatic-javascript。

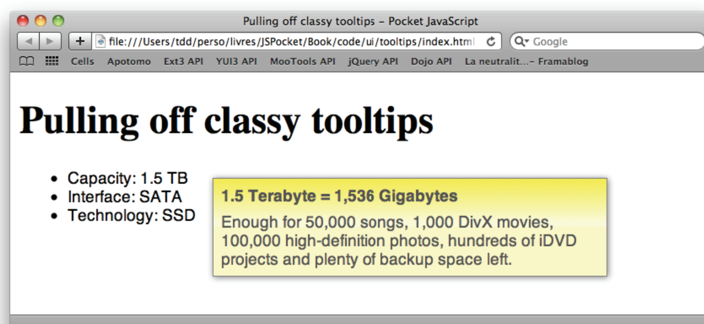
任务 12 打造漂亮的 tooltip

保守地说，HTML自带的tooltip（用 `title=` 实现）并不可靠。自动换行、拉伸与截断之间的相互作用、换行以及显示延迟，这些都随浏览器和用户而变。而且那只是纯文本提示——没有样式，当然也没有标记结构。

为了打造漂亮的tooltip，我们需要自己动手实现：放置一些元素，在鼠标移入或移出页面特定区域时显示或者隐藏它们（或者可以在用户使用Tab切换焦点时显示或隐藏，这样可能更好用）。

由于本书主题是JavaScript，我就不深究CSS代码了。另外，我只做简单的设计：这里的tooltip不会覆盖触发元素的显示、不跟随鼠标移动，也不在里面提供鼠标操作的UI。

所以，根据我们的目的，实现的关键是：用CSS属性设置tooltip元素为默认隐藏，并在其容器标签上（不一定是链接）加`:hover`选择器来恢复其显示，如下图所示。



但这种实现方式在IE6中不起作用，因为IE6只允许 `<a>` 元素上有`:hover`。这时只能手动编写一段脚本以响应`mouseover`和`mouseout`。由于Prototype实现`show()`和`hide()`的方式比较特殊，因此没办法用CSS预先隐藏tooltip——这就是为什么要在右页的CSS样例中针对IE6做一些特殊设置的原因。

如果要实现更高级的需求，你有很大几率能在众多tooltip库中找到合适的——它们要么是直接写在框架里，要么是基于某个框架。就我个人而言，最喜欢的是Prototype的tooltip库——无比强大的Prototip2。①

① 参见http://www.nickstakenburg.com/projects/prototip2/。

tooltip的HTML代码

ui/tooltips/index.html

```
<li tabindex="1">
  <span class="name">Capacity: 1.5 TB</span>
  <div class="tooltip" >
    <p><strong>1.5 Terabyte = 1,536 Gigabytes</strong></p>
    <p>Enough for 50,000 songs, 1,000 DivX movies, 100,000
      high-definition photos, hundreds of iDVD projects and
      plenty of backup space left.</p>
  </div>
</li>
```

设置tooltip样式

ui/tooltips/tooltips.css

```
#files li { position: relative; }
#files li .tooltip {
  position: absolute; top: 8px; left: 120px; width: 24em;
  z-index: 1; display: none;
  /* IE6不识别 li:hover, 所以我们需要用JS触发,
     因此避免使用内置的display: none */
  _display: block;
  border: 1px solid gray;
  background: #fffdc3 url(bg_tooltip.png) top left repeat-x;
}
#files li:hover .tooltip,
#files li:focus .tooltip { display: block; }
```

处理IE6的脚本（IE6中:hover只对链接有效）

ui/tooltips/tooltips.js

```
function toggle(reveal, e) {
  var trigger = e.findElement('li'),
    tooltip = trigger && trigger.down('.tooltip');
  if (!tooltip) return;
  tooltip[reveal ? 'show' : 'hide']();
}

document.observe('dom:loaded', function() {
  var isIE6 = Prototype.Browser.IE &&
    undefined === document.body.style.maxHeight;
  if (!isIE6) return;
  var files = $('files'), tooltips = files && files.select('.tooltip');
  if (!files || 0 == tooltips.length) return;
  tooltips.invoke('hide');
  files.observe('mouseover', toggle.curry(true)).
    observe('mouseout', toggle.curry(false));
});
```

任务 13　制作友好的弹窗

不管用“真”弹窗（单独的窗口）还是“伪”弹窗（实际是当前页面的元素，只是设置成类似窗口的样式），都会碰到这个问题：如何让那些不能（或者不想）打开新窗口的用户访问这些内容？想想窗口弹出被禁用的情况以及屏幕阅读器[①]、搜索引擎的访问等。

只有一种办法：先让链接真正链接到弹窗的内容，然后从那里开始逐步地改善效果。

如果你的内容是要作为HTML片段显示在伪弹窗中（可能还通过Ajax载入），那么不管带不带document标记，你都需要确保在另一处提供其访问路径。那样一来，用户就能通过普通的链接，以单独页面的形式访问到它，看到好的效果。

实际操作的要点是：用`<a>`标签链接到要弹出的内容（`href=`，或许还要加上`target="_blank"`），然后给这些链接挂上JavaScript代码，让它们活动起来。窗口弹出代码很简单——就是用内置的`window.open()`方法。

如果需要伪弹窗或者光箱，一定要使用优秀的现有库以远离地狱般的跨浏览器问题和布局算法。这里有一些可选项：

- Scripty2的UI部分[②]
- jQuery UI[③]
- Dijit（基于Dojo）[④]
- YUI的覆盖模块[⑤]
- Ext.Window[⑥]

另外要记住，如果弹窗内容较少的话，使用单击触发的tooltip就足够了，而不必使用全功能的弹窗。如果遇到这种情况，查查tooltip库和模块。

---

① 屏幕阅读器将屏幕上的文字、图形等转换成语音或盲文，帮助视觉障碍者操作电脑。如果未做仔细的处理，弹窗的内容可能无法被屏幕阅读器访问。*

② 参见http://github.com/madrobby/scripty2。

③ 参见http://jqueryui.com/。

④ 参见http://dojotoolkit.org/projects/dijit。

⑤ 参见 http://developer.yahoo.com/yui/3/overlay/。

⑥ 参见http://www.extjs.com/deploy/dev/docs/?class=Ext.Window。

用于渐进式效果提升的HTML代码

ui/popups/index.html

```
<p>
  The great thing about <a class="popup" target="_blank"
  href="http://pragprog.com/titles/pg_js">Pocket Guide to JavaScript</a>
  is that it focuses on a bunch of specific, useful tasks.</p>
```

普通的**window.open()**脚本

ui/popups/popups.js

```
var POPUP_FEATURES = 'status=yes,resizable=yes,scrollbars=yes,' +
  'width=800,height=500,left=100,top=100';

function hookPopupLink(e) {
  var trigger = e.findElement('a.popup');
  if (!trigger) return;
  e.stop(); trigger.blur();
  var wndName = trigger.readAttribute('target') ||
    ('wnd' + trigger.identify());
  window.open(trigger.href, wndName, POPUP_FEATURES).focus();
}

document.observe('click', hookPopupLink);
```

相关任务

- 任务12
- 任务15

任务 14 预载入图片

页面可能会提供切换图片的用户交互操作，比如切换为放大或特写的图片、前/后视角图片的切换等。这时，由于要用来切换的那张图片需要载入时间，用户会看到瞬间的空白，你肯定不希望做成这样。这就需要你预载入那张要用的图片。

总体来看，只有三种预载入图片的方法。

- **用JavaScript动态创建带有合适`src`属性的`Image`对象：**这种方法允许检测预载入图片何时真的被载入了。
- **用CSS隐藏已载入了的图片：**这其实就是给要预载入的图片使用隐藏的 `<img>` 标签。可以隐藏 `<img>` 标签本身，也可以隐藏这些 `<img>` 标签的公共容器标签（我更倾向于隐藏公共容器标签）。
- **使用CSS sprites：**如果打算预载入rollover[①]这类东西或者一批相关联的图片（比如背景、边框、边角），这无疑是更好的办法。[②]

CSS较为直接，不过其控制能力不及JavaScript。右页的标记和脚本代码做了如下假设：假设每个所在的 `<img>` 标签中带有 `rel="preloadZoom"` 属性的图片在同样的URI下都有一份特写版本，其文件名加有 `_closeup` 后缀。特写版本的图片将被预载入，以实现特写rollover效果。

通常在这种情况下用JavaScript比CSS好。因为rollover功能本身要用JavaScript实现，所以我们就用JavaScript做预载入（这样的话如果用户环境不支持JavaScript，就没必要载入不会被显示的图片了）。此外，用JavaScript预载入可以避免标记及样式膨胀。

如果是用JavaScript，只要我们在确定预载入完成之后再做切换，就能避免出现瞬时空白或不完整显示。而不使用sprite的CSS则有可能会切换到尚未载入的图片（诚然，这样的可能性其实很小）。

想要了解更多关于快速图片载入的核心技巧，请看Steve Souder 2009年5月的演讲稿——快速图片载入的14条法则。[③]

---

① 指一个图片，当鼠标移到上面就变成另一个图片，鼠标移开又变回来。*

② 即“14 rules for faster-loading images”，看看原创的http://www.alistapart.com/articles/sprites和最近的http://css-tricks.com/css-sprites/。Steve Souders同时也维护了一个SpriteMe工具：http://spriteme.org/。

③ 参见http://stevesouders.com/docs/wordcamp-20090530.ppt。

实现预载入功能的HTML代码

ui/preloading/index.html

```
<ul id="products">
  <li><h2><a href="http://pragprog.com/titles/cppsu">
    <img rel="preloadZoom" src="cppsu.jpg" alt="" />
    <span>Prototype and script.aculo.us</span>
  </a></h2></li>
  <li><h2><a href="http://pragprog.com/titles/vsscala">
    <img rel="preloadZoom" src="vsscala.jpg" alt="" />
    <span>Programming Scala</span>
  </a></h2></li>
</ul>
```

实现预载入功能的JavaScript代码

ui/preloading/preloading.js

```
function preloadImages() {
  $$('img[rel="preloadZoom"]').each(function(img) {
    var pimg = new Image();
    pimg.src = img.src.replace(/(\.\w+$)/, '_closeup$1');
  });
}

document.observe('dom:loaded', preloadImages);
```

切换（载入完成的）图片

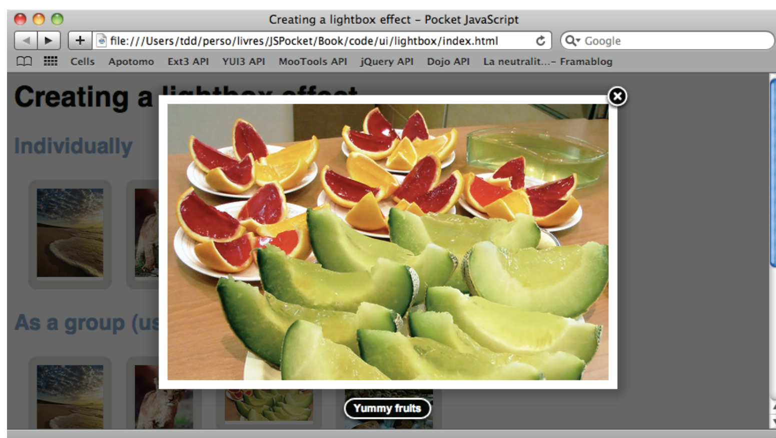
ui/preloading/preloading.js

```
function togglePreloaded(e) {
  var trigger = e.findElement('img[rel="preloadZoom"]');
  if (!trigger) return;
  if (e.type == 'mouseover') {
    trigger.src = trigger.src.replace(/(\.\w+$)/, '_closeup$1');
  } else {
    trigger.src = trigger.src.replace('_closeup', '');
  }
}

document.observe('mouseover', togglePreloaded).
  observe('mouseout', togglePreloaded);
```

任务 15　创造光箱特效

光箱效果指的是突出显示出一部分内容，同时用屏蔽单击的重叠层暗化页面其余内容的操作。光箱一般显示在页面中央，通常是图片，由页面上的文本或缩略图链接触发，如下图所示。



本书的例子中，我们通过Janis Skarnelis的FancyBox jQuery插件来实现把缩略图放大成原始图片的光箱效果。这个插件很友好，比较轻便，也易于插入，而且看上去异常绚丽。但如果你有兴趣的话，也不妨看一下这个demo的页面底部列出的别的实现方式。

FancyBox不只限于显示图片，实际上它能显示任何内容，甚至是`<iframe>`。如果链接的目标URI看上去既“不像图片文件”也没用iframe，它会默认用即时Ajax载入，比如，可以通过不带扩展名的URL来按需载入较大的图片。

和大多数UI相关的库相似，FancyBox需要载入至少各一个JS文件和CSS文件。它会假设自己需要用的图片和CSS文件放在在同一目录下，以免去调整CSS的麻烦。 FancyBox对光箱图片的URL没有要求。

要把链接做成光箱，只需要选取那些链接，然后调用它们的`fancybox()`方法，正如右页的代码所示。既可以不带参数直接调用该方法，也可以传入参数来对一些选项进行设置。我希望看到缩略图动态放大到原始大小，因此就做了相应的设置。

载入FancyBox

ui/lightbox/index.html

```
<link rel="stylesheet" type="text/css"
  href="vendor/fancybox/jquery.fancybox-1.2.6.css" />
<script type="text/javascript" src="vendor/jquery-1.3.2.min.js"></script>
<script type="text/javascript"
  src="vendor/fancybox/jquery.fancybox-1.2.6.pack.js"></script>
```

链接到单个图片

ui/lightbox/index.html

```
<a href="beach_normal.jpg" title="A gorgeous beach">
  <img src="beach.jpg" alt="Beach" />
</a>
```

链接到一组可切换浏览的图片

ui/lightbox/index.html

```
<li>
  <a href="beach_normal.jpg" rel="demo" title="A gorgeous beach">
    <img src="beach.jpg" alt="Beach" />
  </a>
</li>
<li>
  <a href="feline_normal.jpg" rel="demo" title="A cute cub">
    <img src="feline.jpg" alt="Feline" />
  </a>
</li>
```

初始化FancyBox（放了些自定义参数）

ui/lightbox/lightbox.js

```
$(function() {
  $('#thumbnails a').fancybox({
    zoomSpeedIn: 300, zoomOpacity: true, overlayColor: '#000',
    overlayOpacity: 0.6
  });
});
```

相关任务

- 任务12
- 任务13

任务 16 实现“无限翻页”

无限翻页是Gmail给我们带来的概念，它在几个应用场景下的应用宣告了分页导航的末日。许多人（尽管不是所有人）都发现，在浏览大量条目时无限翻页更有效，于是就都倾向于使用视觉模式匹配，而不是移动鼠标去一次又一次地单击分页链接。

分页导航几乎只是基于技术原因提出的。

- 老版浏览器渲染慢，这使我们不得不保持页面“足够轻便”。
- 客户端和服务器之间任何一处都可能有带宽限制，这也要求我们减少页面大小。
- 服务器端的处理时间随着数据增多而增加。一次性渲染整个数据集不仅是浪费，而且对服务器来说也不可行（至少是不现实和笨拙的），尤其是服务器想要处理足够多的并发请求时。

分页导航只不过是一种方法，一种不用JavaScript的方法。启用了JavaScript之后，如果用无限翻页比较合适的话，你就可以不再用任何分页链接，而用无限翻页来代替它们。至于是否采用无限翻页，这就得交给你的用户可用性专家以及你的合理直觉来判断了。

在右页的代码中可以看到，无限翻页并没有什么特别奇特的地方。唯一比较棘手的就是跨浏览器正确计算长度比较麻烦，这从`lowEnough()`方法的代码中可以很明显地看到。

我们实质上是在判断文档底部离当前显示区域底部是否已不太远。这只不过是足够频繁地检查垂直滚动条的当前状态，每秒10次足够了！然后用Ajax载入更多内容。

考虑到可用性（和容错），你或许也需要提供分页导航：把Next链接设置为默认隐藏，如果Ajax获取数据失败就显示这些链接，这样用户还可以回到用分页链接来操作的状态。

给手头没有PHP服务器的读者一个提示：作为demo来说，可以不用访问服务器，客户端直接载入更多内容就能看到效果，这里的PHP代码只不过是在返回静态内容前等待半秒以模拟载入时间。

判断是否已滚动到足够低

ui/infinite/infinite.js

```
function lowEnough() {
  var pageHeight = Math.max(document.body.scrollHeight,
    document.body.offsetHeight);
  var viewportHeight = window.innerHeight ||
    document.documentElement.clientHeight ||
    document.body.clientHeight || 0;
  var scrollHeight = window.pageYOffset ||
    document.documentElement.scrollTop ||
    document.body.scrollTop || 0;
  // 在离页面底端20像素时触发翻页
  return pageHeight - viewportHeight - scrollHeight < 20;
}
```

监测滚动位置并载入更多内容

ui/infinite/infinite.js

```
function checkScroll() {
  if (!lowEnough()) return pollScroll();
  $('spinner').show();
  new Ajax.Updater('posts', 'more.php', {
    method: 'get', insertion: 'bottom',
    onComplete: function() { $('spinner').hide(); },
    onSuccess: pollScroll
  });
}

function pollScroll() { setTimeout(checkScroll, 100); }

pollScroll();
```

任务 17 在载入内容时保持显示区域

有时，你可能希望保持用户的显示区域（也就是在浏览器窗口中可见的所有东西）固定不变，哪怕是用户单击链接以在其上载入更多内容的时候。比如，用户单击查看前面的评论，你不会希望在上面载入评论之后用户的滚动位置就被推到下面去了吧。

这时，你会发现遇到了一点小障碍。感觉上用户应该是期望他们的页面位置没有变化。但是在上面载入内容时，你又必定会把文档的其余内容，包括被单击的那个地方，向显示区域下方推，而且很可能推到了显示区域之外。

解决办法是，保持相对于显示区域的滚动位置。要做到这一点，我们需要获取显示区域的“滚动偏移”，即在当前显示部分上方载入内容前显示区域的滚动位置，然后在载入内容之后恢复这个偏移。计算偏移非常麻烦，到处纠缠着跨浏览器的问题，好在Prototype等JS库提供了位置相关的功能来解决这个问题。

右页的代码改写自Sam Stephenson的Gist[①]。在载入更多内容前，我们先获取触发链接在整个文档中的位置（也就是它的`cumulativeOffset()`），然后从中减去显示区域的当前滚动位置，就得到这个触发链接在显示区域中的出现离区域顶端的距离。新内容载入后，用这个减法的逆过程就能计算出显示区域的新偏移值。

熟悉Prototype的读者也许会奇怪为什么我这里用的是`Ajax.Request`，而没有用到更具体的`Ajax.Updater`。这是因为用`Ajax.Request`可以尽可能地推迟获得原来的滚动位置的时间，即在Ajax请求完成之后再获得。这样即使遇到用户在Ajax请求等待期间滚动了页面，也可以避免出现诡异的“滚动复位”效果。

这样一来，只需通过很少的代码就实现了相当不错的用户体验。

---

① github的代码片段粘贴服务，该Gist位于https://gist.github.com/134653。*

维持显示区域位置

ui/viewport/index.html

```
<h2>Comments</h2>

<div id="extraComments">
  <a id="loadKnownComments" href="?with_known_comments">See previous
    comments you already know about</a>
</div>

<h3>Comment 5</h3>
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
  eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim
  ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut
  aliquip ex ea commodo consequat.</p>
```

在Ajax之前获得滚动位置，在Ajax之后调整[①]

ui/viewport/viewport.js

```
function loadKnownComments(e) {
  e.stop();
  var zone = $('extraComments'), ref = zone.next('h3');
  var upd = new Ajax.Request('known_comments.html', {
    method: 'get',
    onSuccess: function(res) {
      var orig = ref.cumulativeOffset().top -
        document.viewport.getScrollOffsets().top;
      zone.insert({ before: res.responseText });
      window.scrollTo(0, ref.cumulativeOffset().top - orig);
    }
  });
}
```

① 此处代码的意思是“进行Ajax请求，在插入新内容之前获得滚动位置，并在插入之后恢复位置”。*

Part 4 第四部分

表单技巧

表单是现代 Web 应用中的关键。我们经常需要从用户那里采集信息，以填写个人资料、设置任务和服务以及发送信息。然而，现在网上大多数表单的可用性和人类工程学都处于令人遗憾的状态。在这部分，我试着给出几个办法，它们在处理表单时可以改善用户体验（现在越来越多地被称为 UX[1]）。

制作实用表单背后的关键思想是：不要浪费用户的时间。

- 用任务 18 的方法可避免用户做两次提交。
- 让用户知道他们还能输入多少文本，参见任务 19。
- 允许用户批量选择和反选，参见任务 20。
- 用任务 21、任务 22 和任务 23 的方法即时验证尽可能多的用户输入。我想你肯定没听说有谁会喜欢晚一点发现用户名重复或者密码不符合安全性要求吧?
- 给填写特定输入区域提供特殊的帮助，如任务 24 所示。
- 你会在任务 25 中看到怎么做到支持其他类型的输入或自动完成。
- 任务 26 对让用户一次上传多个文件的技巧给出了说明。

以上只是让表单更易于使用的几个例子。后面的任务讲解中我们具体讨论怎么实现这几个例子。

当然，永远不要忘记，你不能认为 JavaScript 理所当然应该能用。不管怎么说，在没有 JavaScript 的情况下，你的表单应该仍然可用。也许它们不那么好用，但必须可用：服务器端无论如何必须再次校验所有内容，在没有 JavaScript 拦截事件时不能出现不起作用的链接等。制作表单时，*渐进式效果提升*至关重要。先保证在没有 JavaScript 或 CSS 支持时它们能正常使用，然后再根据环境中可用和启用了的技术逐层增加用户体验。

① user experience 的特别记法。*

任务 18 暂时禁用提交按钮

有时候，表单要花一点时间等待服务器处理。这有可能是在用老式的`<input type="file".. />`域上传大文件，或者服务器由于某种原因还没来得及响应。无论如何，我们不希望我们还在处理它的时候用户再次提交表单。要知道遇到重复提交是很令人恼火的事。

为了避免重复提交，可以禁用已提交过的表单的所有提交方式。这实际上就是*禁用*带有`type="submit"`或者`type="image"`属性的`<input>`或`<button>`标签。由于有的浏览器（比如“可爱”的MSIE）对CSS属性选择器处理得不好，我们最好还是给这些元素标记上特定`class`属性，比如`submit`，然后再用这个`class`属性选择想禁用的元素。

右页的第一段脚本通过很短的Prototype代码实现了这个功能。它非常简洁。

你可能想进一步给表单加一些等待提交时的UI装饰——因为并非所有的浏览器都清楚地显示被禁用的区域，而且你也可能想要强调*正在处理之中*（你肯定不只是因为那东西讨厌所以才禁用它的吧）。

第二段脚本展示了怎样通过给被禁用的`<input>`标签填加自定义`class`属性来实现等待时的UI装饰。因为用CSS做的UI更新不像禁用函数调用那么实时，所以浏览器在提交开始之前需要一段间歇，否则提交开始之后浏览器就可能一直处理提交事件，而不再响应UI更新请求。因而我们通过`delay()`把`submit()`调用延迟十分之一秒。

另外，注意到这里的JavaScript代码还用到了`that = this`闭包技巧。你也许知道，调用函数（这里是要`delay()`的那个）有可能会丢失当前*束定*，即**`this`**引用的值。一种办法是强制保持束定，这需要一层额外的函数封装，比较麻烦。我们这里用*闭包*来让那个临时定义的匿名函数中的代码维持到原先的**`this`**（被提交的表单对象）的引用，使得在适当的时候能够调用它的`submit()`。[①]

---

① 觉得JS的函数束定很晕吧？在http://www.alistapart.com/articles/getoutbindingsituations/上有我的ALA文章你可以看看。搞不太清楚闭包，也不知道怎么发挥其作用？我的朋友Juriy “Kangax”Zaytsev就这个问题写了一篇很棒的文章，网址在http://msdn.microsoft.com/en-us/scriptjunkie/ff696765.aspx。

发生submit事件时禁用控件

form/submit/submit.js

```
function preventMultipleSubmits() {
  this.select('.submit').invoke('disable');
}

document.observe('dom:loaded', function() {
  $('commentForm').observe('submit', preventMultipleSubmits);
});
```

form/submit/index.html

```
<form id="commentForm" action="post_comment.php">
  <p>
    <label for="edtText">Your text</label>
    <textarea id="edtText" name="text" cols="40" rows="5"></textarea>
  </p>
  <p><input type="submit" class="submit" value="Send" /></p>
</form>
```

用class属性进一步装饰（稍显花哨）

form/submit/submit.js

```
function preventMultipleSubmits(e) {
  if (!this.hasClassName('submitting')) {
    e.stop();
  }
  this.addClassName('submitting').select('.submit').invoke('disable');
  var that = this;
  (function() { that.submit(); }).delay(0.1);
}
```

任务 19 提供输入长度反馈

填写表单时经常遇到的令人沮丧的事就是看到文本输入突然停下——即使有静态文字提示最长长度。不止是这个，你是否知道`<textarea>`没有`maxlength=`属性？真的没有。当然，`<textarea>`不算合法的HTML，可以愉快地忽略掉（除非你有幸能用HTML5）。

因此，为了统一指定最大长度，可以用专门做“数据存储”的CSS `class`属性。这些`class`属性名称由两部分组成：第一部分是`maxLength`前缀，第二部分是用来表示我们想要设定的最大长度的正整数。见右页中的HTML代码。

然后我们用JavaScript进行以下操作。

(1) 动态装饰包含这些元素的表单区域（简单起见，我假设用段落来包含。右页中的JS代码给段落加上了`class`属性），然后动态创建用于显示剩余长度反馈的占位符。

(2) 根据当前输入状态初始化反馈区域。

(3) 为按键事件绑定合适的事件监听器。

(4) 放置反馈区域（我放在对应输入区域右下角下），然后把它加入文档中。现在就能看到效果了！

这样每当发生按键输入时，只需要更新反馈即可。如果达到或者超出最大长度（当然，这在`<textarea>`上不可能发生），就回退到最大允许长度。

另外，注意这份代码中的几个技巧。

- 我们对`keyup`和`keypress`这两个事件都做了监听，以对非字符按键（多半是删除、剪切和粘贴）和字符按键都作出响应。没有必要监听`keydown`，因为它是在文本改变之前发生的，而此时我们没办法跨浏览器和键盘布局判断文本是否会改变。
- 为避免每次按键都重复计算最长长度，我们在初始化时把它缓存起来。为了把最长长度和输入区域关联起来，我们用了JavaScript关联数组，这个关联数组关联了输入区域的`id=`属性[①]和输入区域对象本身。这比用更多的属性要轻便点。

---

① 这里用Prototype的`identify()`来保证我们的元素一定有一个`id=`属性。

用HTML指定最大长度

form/feedback/index.html

```
<p>
  <label for="edtDescription">Description</label>
  <textarea id="edtDescription" name="description" cols="40"
   rows="5" class="maxLength200"></textarea>
</p>
```

为最大长度域绑定反馈事件

form/feedback/feedback.js

```
var maxLengths = {};

function bindMaxLengthFeedbacks() {
  var mlClass, maxLength, feedback;
  $$('*[class^=maxLength]').each(function(field) {
    field.up('p').addClassName('lengthFeedback');
    mlClass = field.className.match(/\bmaxLength(\d+)\b/)[0];
maxLength = parseInt(mlClass.replace(/\D+/g, ''), 10);

    feedback = new Element('span', { 'class': 'feedback' });
    maxLengths[field.identify()] = [maxLength, feedback];
    updateFeedback(field);
    field.observe('keyup', updateFeedback).
      observe('keypress', updateFeedback);

    feedback.clonePosition(field, { setHeight: false,
      offsetTop: field.offsetHeight + 2 });
    field.insert({ after: feedback });
  });
}
```

即时给出反馈

form/feedback/feedback.js

```
function updateFeedback(e) {
  var field = e.tagName ? e : e.element();
  var current = field.getValue().length,
    data = maxLengths[field.id], max = data[0],
    delta = current < max ? max - current : 0;
  data[1].update('Remaining: ' + delta);
  if (current > max) {
    field.setValue(field.getValue().substring(0, max));
  }
}
```

任务 20 同时选择或反选多个 checkbox

常常会遇到这种情况：有一个列表，这上面有用户希望一并完成的操作。要把列表中多项的某个属性一起删除、移动、保存和改变，首先需要让用户选择出特定的项。

有时需要进行全选。在这种情况下，列表长的时候就会让人很难受，所以说批量切换选择状态是不错的UI功能，如下图所示。

☐	Subject	Date	From	Size
☐	Happy new year!	Jan 1, 2010 00:03am	Drew Barrimore	1.6Kb
☐	What's that great IT book publisher again?	Jan 1, 2010 11:25am	David McClintock	2.3Kb
☐	You gotta check this out...	Jan 2, 2010 02:15pm	Julianne Moore	4.2Kb

对应的HTML很简单。在表格的表头放一个checkbox作为切换选框。这自动指定了其作用域，能让我们的脚本代码找到对应的<tbody>，把它作为要处理的那些checkbox的容器。

脚本本身也很清楚：对切换选框的单击作出反应，也就是在对应的<tbody>中找到那些checkbox并更新它们的checked=属性，这样就实现了多个checkbox状态的切换。

现在不妨练习一下，改写这段代码以达到以下两个目标。

- 允许页面中存在多个表格，每个表格都带一个切换选框。这意味着要把id=替换为class属性，然后用事件委托来避免注册过多的监听器。
- 设法处理更复杂的表格，即每个表格中有不只一个<tbody>。是的，这是有效的HTML标记（用某种方式给数据分组时，一个body就是表的一个语义节）。有趣的是，加上不止一个的<tbody>反而会简化脚本代码！

对应的HTML

form/checklist/index_for_book.html

```
<table id="mailbox">
  <thead>
    <tr>
      <th><input type="checkbox" id="toggler" /></th>
      <th>Subject</th>
      <th>Date</th>
      <!-- 格式，大小，附件…… -->
    </tr>
  </thead>
  <tbody>
    <tr>
      <td><input type="checkbox" name="mail_ids[]" value="1" /></td>
      <td>Happy new year!</td>
      <td>Jan 1, 2010 00:03am</td>
      <!-- …… -->
    </tr>
    <!-- 更多行…… -->
  </tbody>
</table>
```

把选择状态传播给每行开头的checkbox

form/checklist/checklist.js

```
function toggleAllCheckboxes() {
  var scope = this.up('table').down('tbody'), boxes = scope &&
    scope.select('tr input[type="checkbox"]:first-of-type');
  var refChecked = this.checked;
  (boxes || []).each(function(box) { box.checked = refChecked; });
}

document.observe('dom:loaded', function() {
  $('toggler').observe('click', toggleAllCheckboxes);
});
```

任务 21　表单验证：基本技巧

客户端验证是必需的。真的。用户的连接吞吐量就不用说了，把本可以直接在浏览器上做的任何验证都传到服务器上绕一圈是非常可耻的。你肯定想让你的网页值得自豪吧，所以让我们开始吧！

第一个任务关注最基本的验证：必填输入项。也就是检验特定输入区域是否不为空或者被选中等。我们约定，用`required`这个CSS class属性标记这种输入区域，之后设置样式来给未填的必填输入项提供视觉反馈。

首先，确保要截取到表单的`submit`事件，而不是submit按钮的`click`事件或者文本域的 Return 按键。在提交到服务器端之前截获表单的唯一可靠方式就是截取它的`submit`事件。它甚至还可以截获到代码引起的提交（`form.submit()`调用）。

一旦挂上submit事件，就只需要获取表单中所有标记为`required`的元素，然后检验它们是否为非空值。用Prototype对String的`blank()`扩展能很方便地检验是否非空。只含空白符的字符串，可以认为是空，因为语义上看它们比实际的空串好不到哪去。然而需要注意，如果输入区域可以接受只含空白符的字符串，要改为用`if(field.getValue())`来判断——在JavaScript中，空串等价于`false`。

右页的JavaScript代码维护一个`firstOffender`引用，这用来自动定位到第一个出问题的输入项，有助于用户矫正输入。另外，做完检验后，如果发现至少有一个问题，就通过`stop()`来终止这个事件，不提交当前表单。

最后，还有一个提示，这算是另一层面的问题：Internet Explorer中的`submit`事件不会冒泡。因此，如果你的代码需要在IE中正常运行，你就得给页面里每个要检验的表单（包括初始DOM载入之后动态插入的）绑定事件监听器。jQuery自1.4版之后在IE中对`submit`事件进行了模拟冒泡，但这是以监视所有表单中的所有单击和按键为代价的，这个代价有点大。

必填输入项的HTML代码

form/validation101/index_for_book.html

```
<form id="registration">
  <p>
    <label for="user_first_name">First name*</label>
    <input type="text" name="user[first_name]" id="user_first_name"
     class="required text" />
  </p>
  <!-- ……更多输入域…… -->
  <p class="radios">
    <input type="checkbox" id="terms" name="terms" class="required" />
    <label for="terms">I accept the terms of service* </label>
  </p>
  <p><input type="submit" value="Sign me up!" /></p>
</form>
```

检测未填的必填项

form/validation101/validation101.js

```
function checkForm(e) {
  var firstOffender, value;
  this.select('.required').each(function(field) {
    value = field.getValue();
    if (value && !value.blank()) {
      field.up('p').removeClassName('missing');
    } else {
      firstOffender = firstOffender || field;
      field.up('p').addClassName('missing');
    }
  });
  if (firstOffender) { e.stop(); firstOffender.focus(); }
}

document.observe('dom:loaded', function() {
  $('registration').observe('submit', checkForm);
});
```

相关任务

- 任务22
- 任务23

任务22 表单验证：进阶技巧

前一个任务检查了必填项是否确实填上或者选中。这很好。不过，我们时常还需要文本域的输入符合某种特定格式，比如电话号码、email地址、整数或者更一般的数字。这些文本域可能已填写，但不符合格式，所以我们要尽可能预先查出来。这通常是对服务器端检查的补充，而服务器端必定会再次做检查。

这一般都用正则表达式[①]来做，它十分清晰有效。如果你做程序开发直到现在还没有习惯正则表达式，最好抽几个小时潜心研究一下，这会对你有很大的帮助。虽然正则表达式在新手看来高深莫测，但是它们只有很少的语法[②]规则（大约有一打，学会其中一半通常就够用了）。学会“正则式”会节省你无数的编码时间。网上有很多不错的交互式教程。

右页代码中的思路是，检测表单项的CSS `class`属性，进行对应的正则式检查。这里我只列了三个正则表达式，你可以在`FIELD_PATTERNS`表里添加更多的键-值对以扩展其功能。为了防止和纯粹主义者[③]多费口舌，我需要先做些解释。是的，这里写的表达式没有覆盖非十进制的数字，小数的科学计数法，以及现在大约百分之0.1的email地址。不过别忘了，这个任务是讲表单域验证，而不是讲正则表达式技巧。随便改这些式子以满足你的需要！

代码很简短明确，我只想阐明两点。第一，`$F(element)`函数其实是`element.getValue()`的缩写。

第二，如果你曾在JavaScript里用过正则表达式的话，你很有可能是一直在用通用的`myString.match(myPattern)`。这是可以的，因为如果没有匹配，它会返回`null`，否则会返回匹配项（或者匹配组）的数组。但当你只想知道是否有匹配，而不关心匹配的具体内容时，就应该反过来问这个问题——让正则表达式来`test()`那个`String`，这只返回`Boolean`值。

实际上`test()`更耐用：即使传给它的不是`String`的值，它也不会出问题。而在不是`String`的值上调用`match()`的话，程序就会崩溃。还有个附带好处就是`test()`运行速度稍微快一点，我自然想利用上这点性能，尤其是在这不会损失代码可读性（或者增加代码量）的情况下。

---

① 一个用来描述或者匹配一系列符合某个语法规则的字符串的单个字符串。在很多文本编辑器或其他工具里，正则表达式通常被用来检索及替换那些符合某个模式的文本内容。许多程序设计语言都支持利用正则表达式进行字符串操作。正则表达式的基本构造元素是：或（|）、重复（*）和优先级（()）。*

② 也有译文法、句法的，指语言的构造规范。*

③ 希望保持一个东西的真正本质，而不掺和杂质或受到稀释的人。*

标记有特定格式要求的输入域

form/validation102/index.html

```
<p>
  <label for="user_email">Email*</label>
  <input type="text" name="user[email]" id="user_email"
   class="required text email" />
</p>
<p>
  <label for="user_favnumber">Favorite number</label>
  <input type="text" name="user[favnumber]" id="user_favnumber"
   class="text number" />
</p>
```

检查特定格式的输入域

form/validation102/validation102.js

```
var FIELD_PATTERNS = {
  integer: /^\d+$/,
  number: /^\d+(?:\.\d+)?$/,
  email: /^[A-Z0-9._%+-]+@(?:[A-Z0-9-]+\.)+[A-Z]{2,6}$/i
};

function checkField(field) {
  var value = $F(field).toString().strip();
  for (var pattern in FIELD_PATTERNS) {
    if (!field.hasClassName(pattern)) continue;
    if (!FIELD_PATTERNS[pattern].test(value)) return false;
  }
  return true;
}
```

相关任务

- 任务21
- 任务23

任务 23 表单验证：高级技巧

前两个任务，即任务21和任务22，检查了必填输入域和输入文本的格式。我们现在要和服务器端通信，做更提前的检查。

这是本书第一次使用Ajax。Ajax是在JavaScript调试中相当烦人的地方。要是你还没有阅读附录B的话，请先读一遍。附录B的信息对你整改事件驱动或Ajax驱动的代码具有不可估量的价值。

Ajax表单验证的典型示例是全局唯一的输入域。通常来说，这处理的是登录名和email地址。我们这里就实现登录名域的这种检查——假设应用场景要求用户之间的登录名不能重复。

Ajax验证的重要特点是即时动态（on-the-fly）UI反馈，比如提示正在检查和提示检查结果，因此HTML标记得要预先留出这些元素的位置。你也需要设计用哪种触发方式，也许会对不同的输入域进行不同的选择。是在输入同时[①]检查（用高频率的输入域轮询）还是在输入完成之后[②]检查（用状态改变事件）？在这个例子中，我用前一种方法实现Prototype的`Field.Observer`机制[③]。我设置了0.8″的时间间隔[④]，这样不至于给打字慢的用户施加太大压力。

我们假定系统要求登录名至少要有两个字符，所以没必要对更短的输入做Ajax查询，这里的检查就忽略掉空串或者单个字符的输入。当有了两个以上字符之后，它才向服务器端检查脚本发送Ajax GET请求，然后根据HTTP响应代码（`2xx = `成功；其他的都表示失败。特别要检查是否有响应状态，因为Opera忽略了许多4xx的返回码）适当地更新反馈UI。

我在服务器端那边放了段短小的伪检查代码。这段代码模拟了网络连接的等待时间，使我们偶尔能看到旋转的等待图标。另外，它根据你输入的登录名是否已知，返回适当的HTTP响应代码。

---

① 在一个输入域输入按键的同时 。*

② 完成一个输入域的文本输入，跳到另一个输入域的时候。*

③ 用法是`new Form.Element.Observer(element, frequency, callback)`，`Field`是`Form.Element`的缩写。*

④ 是的，geeky的朋友，我应该说“1.25 Hz频率”。

登录名域的HTML代码

form/validation_ajax/index.html

```
<p>
  <label for="user_login">Login*</label>
  <input type="text" name="user[login]" id="user_login"
   class="required text" />
  <span class="feedback" style="display: none;"></span>
  <img src="spinner.gif" class="spinner" style="display: none;" />
</p>
```

监测登录名域

form/validation_ajax/validation_ajax.js

```
document.observe('dom:loaded', function checkLogin() {
  var feedback = $('user_login').next('.feedback'),
    spinner = $('user_login').next('.spinner');
  new Field.Observer('user_login', 0.8, function(_, value) {
    if (value.length < 2) return;
    feedback.hide(); spinner.show();
    new Ajax.Request('check_login.php', {
      method: 'get', parameters: { login: value },
      onComplete: function(res) {
        if (Ajax.activeRequestCount > 1) return;
        if (res.request.success() && res.status) {
          feedback.update('Login available!').removeClassName('ko');
        } else {
          feedback.update('Login taken!').addClassName('ko');
        }
        spinner.hide(); feedback.show();
      },
    });
  });
});
```

模拟登录名检查

form/validation_ajax/check_login.php

```
<?php
sleep(rand(5, 10) / 10.0); // 模拟网络间的延迟……
$RESERVED = array('bob', 'doudou', 'tdd', 'meshak', 'ook');
$login = isset($_GET['login']) ? $_GET['login'] : '';
$response = in_array($login, $RESERVED) ? '422 Conflict' : '202 Accepted';
header('HTTP/1.1 ' . $response);
?>
```

任务 24 在表单中提供动态的帮助 tooltip

有时，我们在网上放有复杂的表单。表单域可能有高层语义或者非平凡的输入规则，如下图所示。例如，或许要强调密码域的复杂性要求。这时，最有效（虽然不太礼貌）的做法就是直接写出来警告用户。但如果多个输入域都需要详细的警告和说明，表单就会变得拥挤而零乱。

有一个解决办法。把这些细节放在每个表单域独自的tooltip里，使它们只在需要时（也就是当输入域获得焦点时）出现。简单来说，这是把任务12的做法用到了表单上。

注意，tooltip放置的最佳位置是在label标签里，而不只是在同一<p>标签里。这样如果是盲人用户使用的话，这些信息可以不受屏幕阅读器的当前模式影响，始终能被获取到[1]。

CSS样式，尤其是这些tooltip的位置，对本任务很重要，所以我在下一页给出了部分CSS代码。但这诚然是一个简单的例子。在代码和效果上它都可以变得更复杂：用库来给`focus`和`blur`事件模拟实现弹出效果，就可以不必给每个相关域都手动注册事件监听器。笔者写到这里时，最新的jQuery已经能直接作出渐变弹出效果[2]，你应该能在你喜欢的JavaScript框架找到这个多少有点官方的插件。

---

① 屏幕阅读器一般会根据当前选中的表单输入元素读出关联的`<label>`标签中的文字，而不一定会阅读同一`<p>`标签中的文字。*

② 应该是指jQuery 1.4.1版（2010-1-25发布）中新增的`.live("focus")`和`.live("blur")`。*

把tooltip放在有用的地方

form/tooltips/index.html

```
<p>
  <label for="user_login">
    Login*
    <span class="tooltip" style="display: none;">
      Logins must be unique, at least 3 characters long,
      and may only use letters, numbers, white space,
      hyphens, underscores and periods.
    </span>
  </label>
  <input type="text" id="user_login" name="user[login]"
   class="required text" />
</p>
```

设置一致、简明的外观样式

form/tooltips/tooltips.css

```
#registration label { float: left; width: 6em; position: relative; zoom: 1; }
#registration input.text { width: 14em; }
#registration .tooltip {
  display: block; position: absolute; left: 24em; top: 0;
  padding: 0.35em 0.5em 0.35em 2em; width: 15em;
  border: 1px solid silver;
  color: gray; font-size: 80%;
  background: #ffc url(lightbulb.png) 0.5em 0.3em no-repeat;
}
```

聚焦时显示，失焦时隐藏

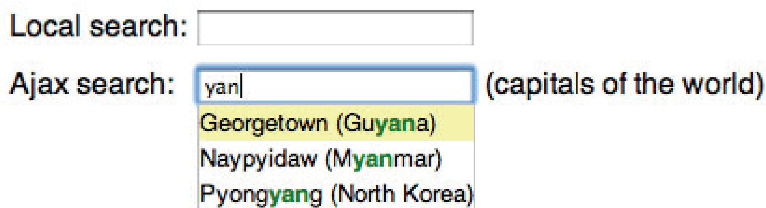
form/tooltips/tooltips.js

```
document.observe('dom:loaded', function() {
  var attr = Prototype.Browser.IE ? 'htmlFor' : 'for';
  function showTooltip() {
    var tooltip = $$('label['+attr+'="'+this.id+'"] .tooltip').first();
    tooltip && tooltip.show();
  }
  function hideTooltip() {
    var tooltip = $$('label['+attr+'="'+this.id+'"] .tooltip').first();
    tooltip && tooltip.hide();
  }

  $('registration').getInputs().invoke('observe', 'focus', showTooltip).
    invoke('observe', 'blur', hideTooltip);
});
```

任务 25 自动完成输入

在表单提交到服务器端之前就验证输入有不错的效果。还有更好的用户体验吗？有！那就是在用户输入的同时做验证。将用户的当前输入和数据库中的有效且合法输入进行匹配，我们可以给用户提供输入建议。这些建议不仅能帮助用户修正拼写等方面的错误，还能自动完成出现在建议列表中的常用项，省去用户手动输入这些文本的麻烦，如下图所示。



传统上，这被称为自动完成。唯一的问题是，你要用的数据源[①]是预先取到客户端（以简单数组的形式，或者用复杂点的结构化的东西，比如JSON字面量），还是就让它放在服务器上，随用户键盘输入不断地去查询。

这里有一条不错的经验法则：如果数据源的数据量足够小（比如是要参考的州名[②]、货币单位或者某个品牌的各种型号名称），那就把它预取到JS里（比如说在初始化页面时给它生成JavaScript字面量），否则就应该用Ajax。使用Ajax需要调好查询频率，即使遇到网速慢或者用户打字快也要保证良好的用户体验。

Script.aculo.us 1.8 有个不错的自动完成控件，它有大量的配置参数。我这里就用它来做自动完成，当然要包括它所基于的Prototype库。根据数据源的位置不同，选择用`Autocompleter.Local`还是用`Ajax.Autocompleter`（我知道这个命名不是很一致，但我也只能对此表示遗憾）。前一种情况需要指定数据源数组，然后用一些参数（`fullSearch`、`partialSearch`、`partialChars`和`ignoreCase`）调整匹配行为；后一种情况则是提供Ajax的基本URI以及那些Ajax相关的配置，包括要发给服务器端的参数。

---

① 用来做匹配的数据。*

② 美国的各个州（state）的名称。*

自动完成的HTML代码

form/autocompletion/index.html

```
<div class="p" id="local">
  <label for="edtCachedSearch">Local search:</label>
  <input type="text" id="edtCachedSearch" name="search" type="text" />
  <div class="completions"></div>
</div>
```

设置可读性更好的样式

form/autocompletion/autocompletion.css

```
.completions {
  border: 1px solid silver; background: white; font-size: 80%; z-index: 2;
}
.completions ul { margin: 0; padding: 0; list-style-type: none; }
.completions li { line-height: 1.5em; white-space: nowrap;
                  overflow: hidden; }
.completions li.selected { background: #ffa; }
.completions strong { color: green; }
```

用客户端里的数据源做自动完成

form/autocompletion/autocompletion.js

```
var FREQUENT_SEARCHES = [
  'JavaScript', 'JavaScript frameworks', 'Prototype', 'jQuery', 'Dojo',
  'MooTools', 'Ext', 'Ext JS', 'script.aculo.us', 'Scripty2', 'Ajax',
  'XHR', '42'
];

function initLocalCompletions() {
  var field = $('edtCachedSearch'), zone = field.next('.completions');
  new Autocompleter.Local(field, zone, FREQUENT_SEARCHES,
                              { fullSearch: true });
}
```

用Ajax做自动完成

form/autocompletion/autocompletion.js

```
function initAjaxCompletions() {
  var field = $('edtAjaxSearch'), zone = field.next('.completions');
  new Ajax.Autocompleter(field, zone, 'autocomplete.php', {
    method: 'get', paramName: 'search' });
}
```

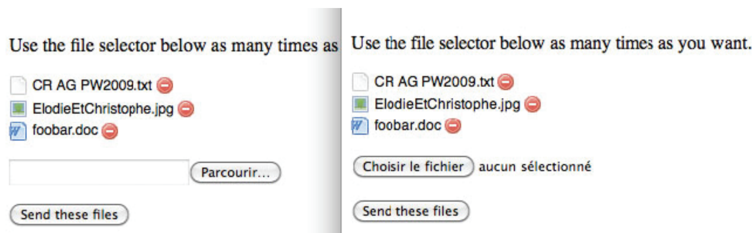
相关任务

- 任务23

任务 26　使用动态多文件上传

HTML（在HTML5之前）里内置的文件上传功能说实话很让人沮丧。它是单文件的，没有上传过程反馈，也不能对文件大小或者文件类型做过滤限制等。而且它使用Base64编码[①]，也就是说每个上传的文件都会膨胀33%。除非我们用WebSockets或者SWFUpload这种东西，否则我们基本上无法摆脱这些限制。

然而，通过给用户提供选择多个文件的好用的操作，我们能提升一点用户体验。我在这里所说的“好用”是指“不是有多少文件就用多少文件上传控件”。我比较喜欢37signals公司在他们的产品[②]中显示待上传文件列表的方式：扁平的、图标装饰的文件名列表，还可以从这个上传“队列”中移除任意项，如下图所示。



秘诀是，每当文件上传控件的值得到设置[③]，就把这个控件复制一份，把原始体控件移到那个“队列”中隐藏起来，并重置控件复制体的值，使其看起来还是空的。下一页的代码通过`<ul>`来实现这个队列，每选择一次文件就合成一个`<li>`来容纳文件上传控件、文件名和移除图标。只是感觉这么做会好一点。接下来，需要对输入进行更多控制（比如一次可以选中多个文件，做到文件大小及类型限制），但用现在的`<input type="file" >`不可能做得到，要实现这些强大的功能，还是用`SWFUpload`吧。

右页中最后那段JS代码用了一个小巧的映射，从而从文件扩展名得到CSS `class`属性名，此外，还实现了当队列中的链接被单击时的队列项移除操作。

---

① 一系列用字符的ASCII码在64以下的字符串来表示二进制数据的编码方法，主要是为了保证数据在传输过程中不会被转义。Base64把每三个8Bit的字节转换为四个6Bit的字节，然后把6Bit再添两位高位0，组成四个8Bit的字节，所以转换后的字符串理论上将要比原来的长1/3。*

② Prototype。*

③ 也就是用那个控件选择了一个文件。*

表单的HTML代码

form/uploads/index.html

```
<form method="post" action="server.php" enctype="multipart/form-data">
  <ul id="uploads"></ul>
  <p><input type="file" name="files[]" id="filSelector" /></p>
  <p><input type="submit" value="Send these files" /></p>
</form>
```

将文件上传项加入队列

form/uploads/uploads.js

```
function queueFile() {
  var fileName = $F(this), clone = this.cloneNode(true);
  var item = new Element('li', { 'class': getFileClass(fileName) });
  $(clone).observe('change', queueFile).setValue('');
  this.parentNode.appendChild(clone);
  item.appendChild(this);
  item.appendChild(document.createTextNode(fileName));
  item.insert('<button><img src="remove.png" alt="Remove" /></button>');
  $('uploads').appendChild(item);
}

document.observe('dom:loaded', function() {
  $('filSelector').observe('change', queueFile);
  $('uploads').observe('click', handleQueueRemoval);
});
```

做得更气派：根据文件扩展名设置样式，移除被单击的队列项

form/uploads/uploads.js

```
var ICONS = $H({ word: $w('doc docx'), image: $w('jpg jpeg gif png') });

function getFileClass(fileName) {
  var ext = (fileName.match(/\.(.+?)$/) || [])[1].toString().toLowerCase();
  var icon = ICONS.detect(function(pair) { return pair[1].include(ext); });
  return (icon || [])[0];
}

function handleQueueRemoval(e) {
  var trigger = e.findElement('button');
  trigger && trigger.up('li').remove();
}
```

Part 5 第五部分

服务器端技术

上一部分我们探讨了如何尽可能多地在客户端校验输入。在做更复杂的校验、自动完成等任务时也开始涉及使用服务器端的数据。大多数 Web 应用都用后端来完成其功能，这一部分的主题就是客户端如何与后端“交谈”。

- 首先谈谈 cookie。cookie 是跨 request 状态持久化的一种早期方式，可以利用它为用户创建浏览过程的 session，以记住用户的操作历史或记住用户。这些都在任务 27 中讲解。存储在用户硬盘上的持久化 cookie 也可以在不同时候的访问之间“记住”用户，这会比较有用。不过很不幸，JavaScript 自带的 cookie 操作不怎么顶用，我们将看到怎样更方便地进行 cookie 处理。
- 然后集中探讨 Web2.0 应用和服务的核心部分：Ajax。我们先讲解在不刷新页面的情况下与服务器端通信的基本操作，这就是任务 28 的内容。
- 接下来，在任务 29 和任务 30 中仔细地探讨一下 JSON。JSON 和 JSON-P 是在运行 JavaScript 的客户端和任何服务器端之间交互数据的好方式（比用 XML 方便多了）。
- 最后，我们讲讲与那些位于不同域名中的第三方服务通信的主要方法。这是任务 31 和任务 32 的内容。

调试 Ajax 或者 JSON-P 调用有时非常棘手。如果你还没有读过附录 B 的话，一定要先去读一遍。在那里，你能找到调试所需的所有工具，这些工具使你能方便地探查到你的代码进行的任何服务器与客户端的交互，因此，完全没必要浪费大量时间为此抓耳挠腮。

任务 27 读取及写入 cookie

在客户端直接操作cookie有不少用处。cookie持久化可以免去用户每次页面刷新后做重复设置，比如重设表单提交和地址重定向之后的分页大小和被激活的选项卡（在选项卡控件中），以及恢复树形控件中的展开/折叠结点状态。

不管用什么方式设置这些cookie（根据作用域或是失效时间），我们能得到的cookie子系统的唯一真正接口是“第0层的DOM”的`document.cookie`属性。这些属性在读的时候充当getter，在写的时候充当setter/deleter。然而很不幸，它并没有进一步为每项单独的cookie设置提供简明的接口。这样写起来感觉就像是在直接读写原始的HTTP头部！

于是，为了避免写起来如此乏味，大多数框架要么直接提供，要么通过某个著名的插件提供了对cookie的更合适的访问方式。然而cookie管理不过是小事一件，没必要靠框架。即使你确实已经在用框架（你也应该用框架），而且这个框架有cookie相关的功能，你也有可能不喜欢它的API。

鉴于这些原因，我在下一页写了个可以单独使用的JavaScript cookie辅助模块。这个模块不依赖于任何框架，试图提供一组便捷的API（尤其在参数方面）。它经过充分的测试而且有丰富的文档说明，也许你会想试一试。

最后，你应该记住关于cookie的几个事实。

- **它们位于客户端，因此处于相当暴露的环境中。**绝不要在那里放一些敏感信息，除非你给它们加了密，并做了坚固的防篡改处理。
- **它们的容量非常有限（4KB）**，不应该用来存储大量数据（比如历史记录、复杂的购物车内容、文本初稿等）。
- **它们可能会不可用，不过这种现象很少见。**稍微常见一点的问题是，由于浏览器的安全策略（可能是因为公司的保密措施，也可能是用户出于隐私考虑而以那种方式配置了浏览器），设置了失效期的cookie在不同session之间丢失了。

考虑到这些情况，请尽量把cookie（尤其是那些持续时间长的cookie）作为Web程序的附加功能来使用。

使用框架或插件

```
// jQuery Cookie 插件（http://code.google.com/p/cookies/）
$.cookies.set(key, value[, options])
$.cookies.get(key)
$.cookies.filter(nameRegExp)
$.cookies.del(key[, options])
$.cookies.test()
// MooTools
Cookie.write(key, value[, options])
Cookie.read(key)
Cookie.dispose(key[, options])
// YUI 2 Cookie 函数
YAHOO.util.Cookie.set(name, value[, options]);
YAHOO.util.Cookie.get(name[, typeOrDecoderCallback]);
YAHOO.util.Cookie.remove(name[, options]);
// YUI >= 3
Y.Cookie.set(name, value[, options]);
Y.Cookie.get(name[, typeOrDecoderCallback]);
Y.Cookie.remove(name[, options]);
// Dojo
dojo.cookie(name, value[, options])
dojo.cookie(name)
dojo.cookie(name, null, { expires: -1 });
// Ext
Ext.util.Cookies.set(name, value[, expires][, path][, domain][, secure])
Ext.util.Cookies.get(name)
Ext.util.Cookies.clear(name)
```

使用我那个可以单独使用的cookie.js辅助模块

```
// 这个辅助模块可以在 http://github.com/tdd/cookies-js-helper 获得
Cookie.get(name)
Cookie.list([nameRegExp])
Cookie.set(name, value[, options])
Cookie.remove(name[, options])
Cookie.test()
```

任务 28　通过 Ajax 载入内容（同域名）

执行Ajax请求，尤其是在当前页面的同域名下，是当前许多Web应用非常基础的构建部分，因而我有必要提及如何在所有主流框架下做这件事。然而，不要说一页，就算是十页也不可能讲清楚所有的细节，毕竟每个框架都提供了大量的Ajax设置选项，需要很大篇幅才能讲清楚怎么设置Ajax行为。

因此在右页只会涉及一些通用的特征，我会在此为你指出正确的方向，提供几条进一步探索的途径。

- 所有框架都允许你用那四个基本的HTTP请求动词（GET、POST、PUT和DELETE）以及调整过的HTTP头部来发送Ajax请求，因此你能够和REST服务等毫无障碍地交互。
- 所有框架都提供了一组回调函数，这样你就能给Ajax响应绑定自定义的处理函数，或者在请求生命周期（开始、成功及失败、完成等）期间做别的事。常见的Ajax响应格式，比如JavaScript、JSON、JSON-P、XML和HTML都有自带的、自动的解码方法（有时还有自动处理的方法，尤其是对JavaScript和JSON-P）。
- 有几个框架允许你为Ajax选项指定通用的默认值（比如jQuery的`$.ajaxDefaults`）并注册全局回调函数（通常是用来维护Ajax指示器，比如著名的“spinner”[①]，它对跨页面的所有Ajax请求都有效）。
- 不少框架都为表单的“Ajax化”提供了快捷的操作方法，要么是直接对它们用HTML强制序列化[②]，要么允许你调整序列化的方式，例如，YUI 3有`form`选项，而Prototype为`<form>`元素加了`request()`操作。
- 每个框架都提供了一些特定功能，比如文件上传、基本的HTTP验证、重进入控制、请求链、响应缓存等。
- 要确保正确处理Ajax调用的成功情况以及失败情况。人们往往都会忽略或者以糟糕的方式处理Ajax失败。

最后一条建议：用同步模式的Ajax基本上等于是在你的网页上召唤邪恶的魔鬼。如果你想实现同步，那就坦率点，用正常的页面重载！

① 在页面上显示出一个不停地动态循环旋转的图标，以标识当前Ajax请求正在进行之中，提示用户耐心等待。*

② 把整个表单及其填写内容编码为一个字符串。一般是对于字段（输入域）比较多的表单，希望通过Ajax动态提交，因为逐项Ajax提交比较麻烦，所以先整个序列化再用Ajax提交序列化得到的字符串。*

进行简单的Ajax请求

```
// Prototype
new Ajax.Request(url[, options])
new Ajax.Updater(container, url[, options])
// jQuery
$.ajax([settings])
// MooTools
new Request([options])
// YUI < 3
YAHOO.util.Connect.asyncRequest(method, url, callback, postData)
// YUI >= 3
Y.io(url[, config])
// Dojo
dojo.xhrGet(settings) // or xhrPost, xhrPut, xhrDelete.
// ExtJS
Ext.Ajax.request(settings)
```

尝试用更细化的、基于Prototype的Ajax

```
Ajax.Responders.register({
  onCreate: function() { $('spinner').show(); },
  onComplete: function() {
    if (0 == Ajax.activeRequestCount)
      $('spinner').hide();
  }
});

new Ajax.Updater({ success: 'latestUsers' }, '/users/latest', {
  method: 'get',
  parameters: { mode: 'summary', threshold: 'auto' },
  evalScripts: true,
  onFailure: function() {
    logError('We could not fetch the latest logged-in users, sorry.');
  }
});
```

任务 29 使用 JSON

在过去几年间，JSON逐渐成为了JavaScript客户端与远程服务器之间交换信息的首选方式。JSON格式[①]实际上是标准的JavaScript中字面量格式的子集。JSON有一定的局限性，但相对于XML，它有两个重要的优点。

- 它更轻便（不那么烦琐）。
- 除了JavaScript自身，它并不依赖于其他任何客户端语言来解释和处理。

现在大多数服务器端语言都可以把合适的数据编码成JSON串，以及对收到的JSON串进行解码。根据你选用的语言，你可能需要安装额外的库（很容易Google到），如果你用的是Ruby、PHP、Python、Java、ColdFusion或者ASP.NET（只是列了几项）的话，可以确信能找到这样的库。

你可以把正常的整数下标数组或者关联数组（也称哈希、字典或者映射）编码成JSON。后者的键（key）可以是任何内容（只要它们是放在引号里），而值（value）可以是数字、字符串（除一些转义外）、布尔值、null或者是任意嵌套的正常/关联数组。JSON解码比较简单，但是生成JSON串需要花一点功夫，所以大多数JavaScript框架都为你实现了这个功能。而且它们通常提供了针对Ajax请求的特定JSON功能，这就更方便了。

注意，JSON应该具有一定的安全性，因此它并不序列化函数。合法的JSON串应该对应只代表数据的对象字面值，当被执行及解释的时候，它自己不会有任何害处。然而，而如果你收到的不是有效的JSON串，而嵌有对恶意函数的调用，那么这就会有危险。正是因为这样，大多数JSON解析功能都提供了在解析之前检查或净化JSON串的选项。

本任务的这份代码实现了用Ajax获取JSON格式的系统信息对象，然后用它生成一个表格，这其中涉及几个简练的技巧。你应该认真看看它！

最近，流行一种使用JSON的方式，叫做JSON-P。它的做法实际上是把JSON对象传给脚本中预定义的回调函数。你将在下个任务中学到关于它的更多内容。

---

① 正如在http://www.json.org/中定义的。

手动解/编码JSON串

```
// 速度快，不过安全性依赖于jsonString
var data = eval('(' + jsonString + ')');

// 自带的 JSON 支持或者 json2.js;，这更安全一些
var data = JSON.parse(jsonString);

JSON.stringify(obj); // 自带的 JSON 支持
```

用框架解/编码JSON串

```
// Prototype带有toJSON()实例方法和evalJSON()方法
Object.toJSON(obj)
someJsonString.evalJSON([sanitize = false])
// jQuery
$.parseJSON(someJsonString)
// Mootools
JSON.decode(someJsonString[, secure = false])
JSON.encode(obj)
// YUI
Y.JSON.parse(someJsonString)
Y.JSON.stringify(obj)
// Dojo
dojo.fromJson(someJsonString)
dojo.toJson(obj[, prettyPrint = false])
// Ext
Ext.util.JSON.decode(someJsonString)
Ext.util.JSON.encode(obj)
```

相关任务

- 任务30

任务 30 使用 JSON-P

要把远程的结构化数据传到JavaScript客户端，最主要的方法就是使用JSON-P。它的传输主要依赖于动态生成的<script>标签，这会产生一个有趣的副作用：传输的数据可以不限于同一来源。现在越来越多的Web服务和API（尤其是REST风格的）都提供了JSON格式的输出和JSON-P支持。

其背后的思想很简单：你得到一份JavaScript代码，这份代码把正常的JSON字面量作为参数传给你提供的回调函数。这意味着回调函数预先由你的代码提供并且设为全局可访问（这样不是很好，但没有别的办法）。

受你的习惯以及你对要用JSON-P的远程资源的信任程度的影响，这要么减轻了你的负担（是一种极好的模仿跨域Ajax的方法），要么需要你做更多的考虑（JSON-P实质上是在你的页面上运行第三方提供的JavaScript）。

如果你只是连接自己的服务器和资源，那就完全没问题，这完美极了。但如果要连接并不完全信任的第三方资源，就会比较麻烦。由于涉及的运行机制（<script>标签），你没办法预解析返回的JavaScript代码以确保它只是做一次安全的JSON-P回调①。如果需要预解析这一中间步骤，你就得先用Ajax获得JS代码，然而正如我们在下个任务中会看到的，对于第三方资源，这种方法并非在所有的浏览器上都可用②。

不管怎么说，最终应该是由你自己来衡量远程JSON-P提供者的可信程度与可靠性。

我必须指出，我们用的传输方式（即动态 <script> 标签）是把JSON-P限制为GET请求，因此也就把加载量限制到了大约4KB（传统的GET限制）。尽管后者没办法纠正，但是只要在服务器端做一些修改，我们就能把GET请求改为其他的请求。我们会在任务32中看到这种做法。

① 期望服务器返回的JS代码是callback(json)，其中callback是客户端设置并告诉服务器端的回调函数名，json是服务器端整合的数据。本例给出的服务器端的这段PHP代码说得很清楚：echo "$cb($response);";。*

② 意思就是要预解析JS代码，还得用跨域Ajax来获取这个JS代码（如果第三方资源是跨域的话），所以比较麻烦。*

通过极简单的代码实现JSON-P

最简单的实现大致是这样的：

server/jsonp/jsonp.js

```
document.documentElement.firstChild.appendChild(
  new Element('script', { type: 'text/javascript',
src: this.href + '&r=' + Math.random() }));
```

上面的代码在一个链接单击事件处理器的上下文中执行，因此this引用的是被单击的那个<a>元素。

这里使用的随机参数是为了避免浏览器缓存而添加的[①]。为了保证代码的质量，最好先检查一下URI，以决定是用 & 还是 ? 作为前缀[②]。

一种更高级的方式是为这种脚本提供动态的 id= 属性[③]，然后在浏览器载入之后将其删除，以免DOM变得过大。这个动态的 id= 属性可以兼作随机参数。

server/jsonp/jsonp.js

```
var script = new Element('script', { type: 'text/javascript',
  src: this.href });
script.src += ('&r=' + script.identify());
script.observe('load', Element.remove.curry(script));
document.documentElement.firstChild.appendChild(script);
```

使用框架中的JSON-P机制

有几个框架为JSON-P提供了专门的函数，这迟早会派得上用场。

```
// jQuery
$.getJSON(url[, data][, callback])
// Mootools
new Request.JSONP({ url: ..., onComplete: function(data) {...} })
// Dojo
dojo.io.script.get({ url: ..., jsonp: function(data) {...} })
```

相关任务

- ❑ 任务29
- ❑ 任务32

---

① 浏览器会缓存一个url的响应数据，如果url完全相同，下一次JSON-P就会自动用之前的数据而不是真的做新的请求。*

② 根据这个参数是否是第一个参数。*

③ 即Prototype的Element.identify()，如果Element当前没有id，调用identify()会为之自动生成一个，形如anonymous_element_x，x从1开始递增。*

任务 31　跨域“Ajax”（方法收集 1）

与第三方服务做数据交互的情况越来越多。在现在这个混搭流行的时代，我们的网页经常需要与服务和内容提供者进行通信，在这个过程中，我们当然希望不要加重服务器的负担。

有不少跨域后台载入数据的办法。我会给出几个比较重要且可靠的方法，它们应该足够对付你的实际需求了。

我先简要地说一下总体情况。

- 任何URL都能用所谓的服务器端代理[①]来载入数据。如果它对你有用的话，那就用它吧。不过在处理文件上传、内容类型（content types）以及POST请求这类东西的时候，可能需要做一些调整（不过一定要检查并过滤到来的请求，你不会希望自己的服务器莫名其妙地为垃圾数据传输打开方便之门吧）。
- 未来进行这种操作的方式将是*跨来源资源共享*（Cross-Origin Resource Sharing，CORS）[②]。XHR2就是用的CORS[③]，这也是W3C指定的跨域数据请求的方式。然而，目前只有Firefox 3.5+、Safari 4+和Chrome支持它。尽管IE8实现了CORS的基本功能（只有GET，不支持自定义头部，不支持证书等），不过它需要你使用一个名为`XDomainRequest`的自定义对象。
- 如果上面两个办法都不可用，那么你就只能使用JSON-P，或者同时使用动态/隐藏表单和`<iframe>`了。[④]
- 目前，还没有特别好的办法能在不用服务器端代理的情况下，就把复杂数据POST到另外的域名。尽管存在一些应对手段（参见下一个任务），但没有一个能够解决全部问题。

右页中是这个任务的代码，其中给出了基于CORS的方法（无需额外代码，只用到`XMLHttpRequest`）、基于服务器端代理的方法以及两个基于表单的动态方法，其中的一个方法用到了`<iframe>`（另一个方法会返回204响应码，使得浏览器不会尝试跳转，所以我们可以不用隐藏的`<iframe>`）。

我必须强调，只有当你不想依赖于某个外部服务[⑤]时，才有必要动用表单和`<iframe>`，否则你就应该选择YQL（这在下一个任务中讨论）。

---

① 将要请求数据的URL发给自己的服务器端，由自己的服务器端获取这个URL的数据之后再返回（比如用PHP的`readfile($uri);`）。*

② 意思是实现了CORS的新版浏览器本身就能做跨域的Ajax请求。*

③ 关于CORS和XHR2的更多信息，请访问https://developer.mozilla.org/en/HTTP_access_control。

④ 隐藏表单 + `<iframe>`一般用于数据提交（还是得放在url的参数里>_<），如果要把返回的数据传给父页面会遇到iframe的权限问题，因为iframe除了src属性可写之外也是不能跨域。当然也不排除有一种极端的手段：在前面设置的iframe A里再嵌套一个和A的父页面同域名的iframe B，把得到的返回数据放在B的src属性的参数（?后面）或者标签（#后面）中，这样就可以由B把这些数据传给A的父页面了。*

⑤ 指YQL服务。如果通过YQL跨域的话系统的正常运行就依赖于YQL的正常运行。*

使用CORS兼容的XMLHttpRequest

server/crossdomain1/crossdomain1.js

```
new Ajax.Updater({ success: 'responses' }, this.href, {
  method: 'get', insertion: 'bottom'
});
```

使用服务器端协议

server/crossdomain1/crossdomain1.js

```
new Ajax.Updater({ success: 'responses' }, 'ssp.php', {
  method: 'get', parameters: { uri: this.href }, insertion: 'bottom'
});
```

同时使用动态生成的表单和<iframe>

server/crossdomain1/crossdomain1.js

```
var warp = new Element('iframe', { name: '__blackhole' });
warp.setStyle('width: 0; height: 0; border: 0');
document.body.appendChild(warp);
warp.observe('load', function() {
  $('responses').insert('<p>OK, posted.</p>');
});
var form = new Element('form', { method: 'post', action: this.href,
  target: '__blackhole' });
form.submit();
```

使用得到204响应的动态生成的表单

server/crossdomain1/crossdomain1.js

```
var form = new Element('form', { method: 'post', action: this.href });
form.submit();
Element.insert.defer('responses', '<p>OK, posted.</p>');
```

相关任务

- 任务32

任务 32 跨域“Ajax”（方法收集 2）

除了前一个任务中的方法，还有几种后台访问其他域名的远程内容的办法。初学者可以用可靠的JSON-P。有一种JSON-P的有趣用法可用来访问大量的服务、API和内容，这就是Yahoo!提供的YQL服务。YQL允许你读（有时还可以写）一些数据表，它们可以映射到你能想象的所有可能的资源：著名的网站和各种服务（包括搜索、地图、地理定位、社交网络、Flickr、音乐库、天气、feed①、微格式②等）。要是你还没玩过这些的话，现在赶紧去看看吧。③

YQL提供两个巨大的“数据库表”，分别叫做html和htmlpost（后者是由Chris Heilmann提供的社区维护的表，因此它完胜前者），可以任意访问资源，然后得到其原始HTML响应。它们允许你把数据GET或者POST到一个返回HTML的资源，甚至还可以通过XPath④选择器把内容提取出来。

此外，还有另一种非常好的办法——`CSSHttpRequest`。不过，它需要服务器端的配合。这得靠`data`：可以先把内容放在URI Scheme⑤里，然后再放在特定名称的CSS规则中。因为CSS文件不受同源策略的限制，所以这能管用。有个很小的开源库提供了`CSSHttpRequest`这个JavaScript对象，以及Ruby、Python和PHP的服务器端代码（很容易移植到其他语言中）⑥。你可以通过网络获取更多细节⑦。

你可以用本书代码库中这个任务的示例代码来试试上面提到的所有远程访问的办法。

另外，再概述一下两种我有意没放在前面的办法，这样你就不会说我忘了它们。第一个方法是使用Flash桥。虽然这种方法需要服务器端提供某种形式的CORS信息，但有一个称为flXHR的库，使得这个操作变得相当容易。不过，Flash正巧是我不怎么喜欢的私有技术，就算不考虑这一点，仅仅为跨域远程访问就载入Flash也可以说是“杀鸡用牛刀”了。

最后，你也可以利用一个“Web的bug”，即利用普通的`<img>`标签（把你的脚本代码放在里面）在动态选中的图片上进行服务器端调用。不过问题在于，你只能基于图片的长和宽这两个数来传递响应信息，而且要在图片载入之后检测出来的图片大小是相当麻烦的。考虑到其响应数据如此有限，还得要克服那么多的困难，我就不用这个办法了。

---

① 网站提供更新信息的接口，常见的feed格式有RSS、Atom两种。详见wiki: http://zh.wikipedia.org/wiki/消息来源。中文译名有消息源、订阅源、网源等。本译文保留原单词。*

② 一种对Web数据添加人机可读的语义标记的开放方法。详见wiki: http://zh.wikipedia.org/wiki/微格式。*

③ 参见http://developer.yahoo.com/yql/。

④ XML路径语言（XML Path Language），用“/A/B/C”这种写法来定位XML文档结点。*

⑤ URI Scheme是指URI的格式，写法是 <scheme name> : <hierarchical part> [? <query>] [# <fragment>]，比如在http://163.com/ 这个URI中，http属于<scheme name>，//163.com/ 属于< hierarchical part >。*

⑥ 客户端只需调用`CSSHttpRequest.get(url, callback)`，服务器端返回注6中那样的CSS编码，之后`<data>`会传给`callback`这个函数。*

⑦ 参见http://nb.io/hacks/csshttprequest。

使用之前那种普通的JSON-P

```
window.jsonpCallback = function jsonpCallback(data) {
  $('responses').update(data.payload.escapeHTML());
};
document.documentElement.firstChild.appendChild(
  new Element('script', { type: 'text/javascript',
src: this.href + '?r=' + Math.random() + '&callback=jsonpCallback' }));
```

使用JSON-P-X[①]形式的YQL html表

```
function yqlCallback(data) {
  // data.results是匹配元素HTML片段的数组
};
var url = "http://github.com/languages/Ruby/updated",
  xpath = "//*[@class='title']",
  yql = 'select * from html where url="'+url+'" and xpath="'+xpath+'"',
  data = { q: yql, format: 'xml', callback: 'yqlCallback' };
document.documentElement.firstChild.appendChild(
  new Element('script', { type: 'text/javascript',
    src: 'http://query.yahooapis.com/v1/public/yql?' +
      Object.toQueryString(data) + '&r=' + Math.random() }));
```

使用JSON-P形式的YQL htmlpost表

```
function yqlCallback(data) {
  // data.query.results.postresult.p == 匹配元素内容的数组
};
var yql = 'use "http://datatables.org/data/htmlpost.xml" as htmlpost;\
    select * from htmlpost\
    where url="http://demos.pocketjavascript.com/server/jsonp/postdemo.php"\
      and postdata="foo=foo&bar=bar" and xpath="//p"',
    data = { q: yql, format: 'json', callback: 'yqlCallback' };
document.documentElement.firstChild.appendChild(
  new Element('script', { type: 'text/javascript',
    src: 'http://query.yahooapis.com/v1/public/yql?' +
      Object.toQueryString(data) + '&r=' + Math.random() }));
```

使用CSSHttpRequest

```
CSSHttpRequest.get(this.href, function(res) {
  $('responses').insert('<p>' + res.escapeHTML() + '</p>');
});
```

相关任务

- 任务30
- 任务31

---

① XML格式的JSON-P。*

Part 6 第六部分

使用混搭

本书的最后一部分包含几个具体例子，它们讲的是怎么制作放在你自己的域名里并且连接第三方服务的网页。这里没有用到任何自定义服务器端协议或者 XML，只用到了 JSON-P 和 YQL，所以这些代码自己就能运行，并且可以在任何地方单独使用！

然而，受篇幅所限，本书的代码页只包含代码的关键部分，而且部分代码可能需要一定的封装调整。因此务必要查阅源码包[①]或者在线 demo 网页[②]，以理解代码的运行机制。

- 把 Twitter feed 同步到网页上是如今经常要用到的功能。最好在客户端实现这个功能，否则会给服务器端带来沉重的缓存压力。任务 33 将为你解释这一点。
- 把你最新在 Flickr 上传的照片同步到某处的一块页面同样常会用到，这在任务 34 中作解释。
- 第三个任务，任务 35，探讨了网络中越来越重要的一个方面：地理编码（不要和地理定位相混淆，尽管它也是一个重要的新趋势，但它只能在最新的前沿浏览器中用）。本质上来说，地理编码把照片的位置名和地址变成了实际的地理坐标，使得你能把这些位置名和地址“钉”在地图上，并和其他数据建立起关系（比如来自 Flickr 或者 tweets 的照片），还可以创造出把多种数据集绑定在一起的各种有用的混搭。如今网络和其中的数据比以往任何时候都更容易获取，因此我们可以利用它们写出无穷无尽的应用！

① 参见 http://pragprog.com/titles/pg_js。
② 参见 http://demos.pocketjavascript.com/。

任务 33 Twitter 的同步更新

获取你最近的tweet可以说是轻而易举，就和使用大多数Twitter的API一样简单。实际上，这只是简单的JSON-P调用罢了。

这里不去获取retweet[①]、mention[②]这些内容。因为一般说来，大多数的Twitter同步发生在商业环境下，而这种情况下，你的Twitter账户是被用作一种额外的营销渠道，所以在自己的信息旁边显示retweet、reply[③]或mention这些信息的意义就不大了。此外，Twitter API并没有提供一种直接在取得tweet同时一并获取retweet的办法，因为，这样一来，你得做两次调用，其中一次需要用户认证，这就意味着你的访问者的浏览器对应的客户端能够获得你自己账户的认证。我们当然不想弄成这样（还有一个选择是在服务器端做同步，不过这就超出本书讨论的范围了）。

在右页的代码中，实际获取数据的代码位于`loadTwitterStream()`，只占短短几行。Twitter允许你通过URL读取任何用户的tweet[④]，我们这里只对JSON格式的信息感兴趣。

你取回的是tweet对象的数组，每个对象都有丰富的属性，比如`created_at`、`geo`、`in_reply_to_status_id`、`source`和`text`。[⑤]

右页中的`twitterCallback()`函数阐明了如何做简单的tweet格式调整：给reply、mention和URL全部赋予超链接。这个任务的在线示例版本的代码是一个略有改动的版本，它还可以处理hashtag[⑥]和mention，并显示作者信息（头像、姓名、tweet总数统计等）。

在用Twitter的API做更多工作之前，你有必要浏览一遍Twitter API文档[⑦]。另外，要注意部分API上带有一定的访问频率限制，以避免你的过度使用超出了Twitter的系统负荷。

---

① 在twitter中对别人消息的转发，用RT或者Retweet标签标识，也称回推、锐推、转推。*

② 在twitter消息中，用@符号后跟某个用户名（也就是提到那个用户），使得那个用户能在其reply监控页面中直接看到此消息。*

③ 对别人消息的回复，公开回复也是用@符号，私下回复就是发送站内消息。*

④ 参见http://twitter.com/statuses/user_timeline/username.format。

⑤ 详见http://apiwiki.twitter.com/Return-Values。

⑥ twitter的#标签，用于给消息指定一种类别，使得同一类别的消息能够聚在一起查看。*

⑦ 参见http://apiwiki.twitter.com/。

获取最新的tweet

下面的代码用到了几个Prototype的函数（$()、each()、escapeHTML()、insert()等），不过比较容易转换到其他框架下。

mashups/twitter/twitter.js

```
var REGEXP_URL = new RegExp('(https?://.*?)(\\W?(?:\\s|$))', 'gi');

function twitterCallback(data) {
  var stream = $('twitterStream'), replyTo, contents;
  data.each(function(tweet) {
    contents = tweet.text.escapeHTML().replace(REGEXP_URL,
      '<a href="$1">$1</a>$2');
    if (replyTo = tweet.in_reply_to_screen_name) { // Intentional assign
      contents = contents.replace('@' + replyTo,
        '<a href="http://twitter.com/' + replyTo + '/statuses/' +
        tweet.in_reply_to_status_id + '">$&</a>');
    }
    contents = '<li><p>' + contents + '</p>' +
      '<p class="stamp">' + tweet.created_at + '</p></li>';
    stream.insert(contents);
  });
}

function loadTwitterStream(userName) {
  var uri = 'http://twitter.com/statuses/user_timeline/'+userName+'.json';
  document.documentElement.firstChild.appendChild(
    new Element('script', { type: 'text/javascript',
      src: uri + '?callback=twitterCallback&r=' + Math.random() }));
}
```

看看tweet JSON编码的一部分

（实际返回的数据要详细得多，而且显然URL不会缩写，这里只是给出了大致的情形罢了。）

```
{
  "in_reply_to_screen_name": null,
  "user": {
    "friends_count": 27, "statuses_count": 622,
    "name": "ChristophePorteneuve",
    "followers_count": 215,
    "profile_image_url": "http://a3.twimg.com/.../headshot_tdd_normal.jpg",
  },
  "id": 9537162839, "created_at": "Tue Feb 23 18:35:22 +0000 2010",
  "in_reply_to_status_id": null,
  "text": "15' pour 850m. Sympa av Saint-Ouen + av Clichy aux heures de..."
}
```

任务 34 Flickr 的同步更新

尽管Flickr提供了相当多的与REST兼容的API，不过它有用的那些API大多需要用户认证。这比较麻烦，至少不是很适合要公开同步的内容。

可以用YQL来完成大部分有用的查询功能，而不需要认证。但是在本任务的特殊情况下，我们用Atom feed的JSON数据变体[1]来做也是可行的，这是Flickr给大部分页面（包括Flickr自己的用户页面）提供的API。得到的结果数据集包含我们需要的所有信息，包括预置图片和图片描述的URL、图片长宽和发布日期。虽然这个feed仅限于提供最近的二十个图片更新，但它恰好就适合我们这里的“Flickr更新”同步。

右页的JSON片段就是我们会得到的那种feed响应数据。在这里我们只对照片缩略图的URL和照片发布日期最感兴趣。再多做一些工作的话，还可以得到照片的原始长宽以及拍照日期。不过为了简单起见，我们还是只单独请求一次，不做太多调整。

这里我们得到的图像URL，它指向的目标是照片的中等大小版本。如果要用更小的正方形缩略图，把图像URL的后缀从 `_m` 改为`_s` 即可。

右页的代码还很好地阐明了Prototype的`Template`类的用法，这允许我们以一种可复用的方式高效地计算出“格式化的字符串”。

还是那句话，用YQL给Flickr提供的表可以在未认证的情况下得到更多信息（先请求一次得到简略的初始响应结果，再分别查询每个照片以获得更多详细资料）。不过，如果你需要进一步获取更详细的信息，或者干脆要更新数据的话，还是使用经过用户认证的Flickr API吧。

---

① 指对Atom feed的JSON做过一些调整的JSON数据。*

获取某人公开的照片

mashups/flickr/flickr.js

```
var FLICKR_ENDPOINT='http://api.flickr.com/services/feeds/photos_public.gne';
var FLICKR_USER_ID ='97027332@N00'; // 这是作者的ID
var item = new Template(
  '<li><a href="#{target}"><img src="#{src}" title="#{title}" /></a></li>');

function jsonFlickrFeed(data) {
  var stream = $('flickrStream'), d, dateStr;
  data.items.each(function(photo) {
    d = photo.published.split(/\D/);
    dateStr = d[1] + '/' + d[2] + '/' + d[0];
    stream.insert(item.evaluate({
      src: photo.media.m.replace('_m', '_s'), target: photo.link,
      title: 'Published on ' + dateStr + ' GMT'
    }));
  });
  $('indicator').removeClassName('loading').update('Loaded!');
}

function loadFlickrPhotostream() {
  var uri = FLICKR_ENDPOINT + '?format=json&id=' + FLICKR_USER_ID;
  document.documentElement.firstChild.appendChild(
    new Element('script', { type: 'text/javascript',
      src: uri + '&r=' + Math.random() }));
}
```

JSON-P响应的部分内容

```
jsonFlickrFeed({
  // ...
  "items": [
  {
   "title": "P1010071",
   "link": "http://www.flickr.com/photos/97027332@N00/4105961623/",
   "media": {
     "m":"http://farm3.static.flickr.com/2638/4105961623_ec0ca9c164_m.jpg"
   },
   "date_taken": "2009-11-12T15:38:21-08:00",
   // ...
   "published": "2009-11-15T18:54:21Z",
   // ...
  },
  // ...
  ]
})
```

任务 35 获得地理位置及该位置的照片

主要由于移动互联网的兴起，地理定位现在成了比较普遍的需求，因此让我们来看看它的两个主要方面。

- 第一，将文本位置（地址、城市、行政区或者州、国家）转换成地理位置（实质上就是经纬度）[①]；
- 第二，用这些地理坐标来搜索数据。

有不少API可以选用，包括Google、Yahoo!、Geonames还有一些其他人开发的著名的“地理编码器”。根据我们的需要，我就用一个简单直接且实用的工具：Yahoo! 的Placemaker API，再加上Christian Heilmann为它打造的JavaScript wrapper[②]，即JS-Placemake。这个API允许我们分析任何文本，并提取该文本可能对应的一个或多个地理位置。我们就用它来把位置名称（比如在某个表单域中输入的文本）转换成经纬度坐标。

和所有其他Yahoo!开发者网络（Yahoo! Developer Network）的API一样，Placemaker需要Yahoo! AppID。右页的示例代码中有个能用的AppID，不过你最好自己申请一个来玩。[③]接着，你要做的就是把这个AppID告诉那个JavaScript wrapper，之后调用它的`getPlaces()`函数，提供待分析的文本、回调函数（用来处理结果）和可选的大体位置（例如en-US或者fr-FR，用来帮助Placemaker做正确的分析）作为这个函数的参数。

注意到代码里用来把结果确定地“正规化”为位置数组的小技巧。由于存在单个匹配时只返回一个单独的`match`属性，而存在多个匹配时则会返回名为`matches`的一个数组，我们采用`(matches || [match])`这个构造来把这两种情况的返回值统一转换成数组来访问。

在涉及Flickr这边，我们只是通过一个与上个任务中类似的JSON-P调用，用地理相关参数`lat`和`lon`调用 flickr.photos.search方法。由于我们是做全局搜索，这次调用无需指定用户ID。

如果你对客户端上的和基于JavaScript的其他地理相关技巧感兴趣，比如GeoIP以及W3C Geo API，你可以在由非凡的福音传道者[④]Christian Heilmann维护的页面[⑤]上找到丰富的信息、demo以及另一些很酷的东西。

---

① 有时你也会想得到额外的数据，比如精确度和结果包围区域。

② Wrapper一般指对现有API的封装，目的是使之更简洁易用。*

③ 在https://developer.apps.yahoo.com/wsregapp/这里申请AppID。

④ 这是对于真心崇拜地理定位的人来说的，那些不看重地理定位的人和更多从未听说过地理定位的人不会这么想。*

⑤ 参见http://isithackday.com/hacks/geo/。

获取给定文本的地理位置

(以下代码实际上假设只有第一个地理位置结果有用。)

mashups/geo/geo.js

```
// 在你自己的代码中用你自己的API码 :-)
var YAHOO_APPID = 'KwWEZW_V34GVYNWWOLZm6NT.' +
  'XfIwNrF9ysko8qu6sDuE6SbehuptUZQp6jKF130V25hFTMrrdrbQeo4-';

function getGeoLocationFor(text) {
  Placemaker.config.appID = YAHOO_APPID;
  $('indicator').addClassName('loading').update('Getting geolocation for ' +
    text.escapeHTML() + '...').show();
  Placemaker.getPlaces(text, function(places) {
    if (places.error) {
      $('indicator').removeClassName('loading').
        update(places.error.escapeHTML());
    } else {
      var loc = (places.matches || [places.match])[0].place;
      $('indicator').update('Loading ' + loc.name +
        ' pics (' + loc.type + ')...');
      getGeoPhotos(loc.centroid.latitude, loc.centroid.longitude);
    }
  }, 'en-US');
}
```

从Flickr获取指定地理位置的照片

mashups/geo/geo.js

```
// flickrCallback非常类似于Flickr同步任务的代码
// 在http://pragprog.com/titles/pg-js/source_code这个网址上获得完整的示例代码

function getGeoPhotos(lat, lon) {
  $('indicator').addClassName('loading').show();
  var uri = FLICKR_ENDPOINT + '?' + Object.toQueryString({
    method: 'flickr.photos.search', api_key: FLICKR_API_KEY,
    extras: 'date_taken,url_sq,description', lat: lat, lon: lon,
    per_page: 50, format: 'json', jsoncallback: 'flickrCallback'
  });
  document.documentElement.firstChild.appendChild(
    new Element('script', { type: 'text/javascript',
      src: uri + '&amp;r=' + Math.random() }));
}
```

Part 7

第七部分

附　　录

附录A　JavaScript快速参考

内置类型和字面量

Number	0，52，0.15，3e10，-3.12e-2
String	"Hello"，'hello'（均可使用转义序列）；常用转义序列：\r \n \t \" \' \uXXXX（十六进制）\0ooo（八进制）
Boolean	true，false
Array	[]，[1，2，3]，[[[3]，2]，1]等
Date	new Date(...)（无字面表示法）
RegexExp	/pattern/flags（详见下文）
Function	function(...){...}（详见下文）
Object	{ prop: value, prop2: value2... }

typeof 表达式 → 类型名称, 小写

将0、''（空字符串）、null和undefined转换为布尔类型时会得到false，转换其他值均会得到true。

正则表达式回顾

类别（匹配字符的集合）	. \d \D \w \W \s \S [...] [^...] （必须以字面连字号结尾）
边界	\A \Z ^ $ \b \B
贪婪匹配符（最大匹配）	* (0+) ? (0-1) + (1+) {min,} {,max} {min,max}
懒惰匹配符（最小匹配）	*? ?? +?
分组	(...)捕获分组（可作为后向引用），(?:...)不捕获分组（速度更快）
前向匹配	(?=...) 要求字符串前的内容被匹配，(?!...)需要字符串前的内容不被匹配
后向引用	\1...\9（代表组的序号），\&（整个匹配串）
或	|（如果在一个组中出现，则仅作用于此组内的内容）

标识位

g	全局匹配：匹配所有的目标字符串，而非第一次出现
i	大小写敏感（对Unicode也适用）
m	多行匹配：点字符（.）可匹配任意字符，包括换行符

- 多次编译同一个正则表达式会造成性能损耗，尽量对其进行预编译。
- 如果只想检查是否存在匹配而不关心其他细节（例如组）的话，使用regex.test(str)而不是str.match(regex)。
- 如果不需要捕获组的内容作为后向引用，而仅仅是在组上应用一个限定符，请把这个组标记为非捕获型：(?:...)。

函数

- 声明：function fxName(...) {...}
- 表达式：function(...) {...}

声明被“提升”到它们的作用域中：作用域中的每一个函数都可以调用其他函数，而不受它们的声明顺序影响（这里并不需要“原型声明[①]”）。

- "return x;" 退出函数并返回x。
- "return;" 退出函数返回undefined。

函数本身也是一种对象，所以函数拥有自己的方法（比如apply和call），也可以按引用被传递（但这样你就会丢失它们的绑定关系：this关键字原有的含义）。这意味着你可以对函数赋值，或者将函数作为参数来传递。

函数的参数列表中并不限定参数类型，参数名起到的更多是提示作用。JavaScript会把实参赋到对应的形参上。需要注意的是，所有的JavaScript参数都像varargs[②]一样：你可以传递任意数量的参数，并通过内置的参数变量表，以访问数组的形式访问它们。

特殊值

undefined	未被赋值的变量，未匹配实参的形参
null	没有被明确的定义；一般是DOM调用“无结果”的返回值，一般在程序中被作为“空值”赋给某个变量
false, true	两个布尔值字面量
Infinity	无限的数值（可以带有正负号，比如-Infinity）
NaN	非数值（Not-a-Number）表达式的返回值，比如说失败的数值解析。它与任何数值不等，包括它自己，所以一般通过isNaN(n)来判断变量是否为NaN

① C/C++ 语言中在使用一个函数前，必须对其进行原型声明，否则无法使用。*

② C语言中为支持可变参数而采用的机制。*

操作符

算术操作符：+ - * / % ++ --

注意：JavaScript中没有int或是float类型，只有Number类型。

- % = 求模/取余。负号（一元操作符）会改变数值符号：-42。
- ++ 和 -- 分别为自增和自减。优先使用前置符（++x），除非有明确的原因要使用后置符（x++）。

比较操作符：== === < > <= >= !=

- == 仅对值进行比较，并使用一套“转换协议”，所以会有 5 == "5"、0 == false、null == undefined等。
- === 进行严格的比较，既对值作比较，也对类型作比较。undefined === x 非常适合严格的测试。
- != 不等号（类似于 VB/SQL’s 中的 <>）。

布尔逻辑操作符：&&（逻辑与） ||（逻辑或）

注意：

- &&比||的优先级高。
- 如果a为真，a || b 会把b短路（即不去求值）。
- 如果a为假，a && b 会把b短路。

算术操作简写符：+= -= *= /= %=

a += b 等价于 a = a + b，其余类似。这样的写法更简洁。

成员选择符：[...]

- 数组通过下标来选择元素：array[5]（0作为起始下标）。
- 动态成员选择：
  - obj['propName'] 等价于 obj.propName；
  - obj['methodName'](...) 等价于 obj.methodName(...)；
  - 例如node[visible? 'show' : 'hide']()。

优先级：和C/C++/Java/Ruby中的操作符优先级类似。

- * / % 优先级高于 + -

- `&&` 优先级高于 `||`
- `(...)` 强制改变优先级
- `(5 * 3) + (6 / 2)` 等价于 `5 * 3 + 6 / 2`
- `(a && b) || (c && d)` 等价于 `a && b || c && d`

位操作符（很少用到）：`& | ^ << >> >>>`

`^` 是异或操作符，`<<` 和 `>>` 为右移位和左移位操作符（`>>>`为无符号右移操作符）。

变量和作用域

```
var x, y=42, z, t=y*2;
```

局部变量必须通过**var**关键字声明。

尽可能不要使用全局变量。

变量可以在声明时被赋值。它们可以在其被声明的区域内被访问（大多数情况下是当前函数体）。

如果你的代码需要隐藏一些标识符（变量或者是函数），请使用**模块模式**[1]：

```
(function(){
  //这里的内容不会被外界所访问到
  //除非你返回里面的成员或直接对里面的成员赋值
})();
```

分支判断语句

```
if(condition){
  code
}

if(condA){
  codeA
} else if (condB){
  codeB
} else {
  codeC
}
```

```
switch(expr){
  case valueA:
    codeA
    break;
  case valB1:
  case valB2:
    codeB
    break;
  default:
    codeC
}
```

提示：尽早返回值，以免陷入到层层嵌套的`else`代码块中，同时请保持代码的缩进正常，以便阅读。

---

① 详见任务2（通过模块模式实现代码访问控制）。*

应该了解的数组知识

```
var arr=[2, 3, 4, 5, 6];

arr.length                  // => 5
arr.concat([7, 8, 9])       // => [2..9]
arr                         // => [2..9]
arr.push(10)                // => [2..10]
arr.unshift(1)              // => [1..10]
arr.pop()                   // => 10
arr.shift()                 // => 1
arr                         // => [2..9]
arr.slice(4, -2)            // => [6, 7, 8]
arr.join('')                // => '23456789'

arr.sort(function(a, b){
  return b - a;
}) // => [9, 8, 7, 6, 5, 4, 3, 2]

arr.sort()                  // => [2, 3, 4, 5, 6, 7, 8, 9]
arr.splice(5)               // => [7, 8, 9]
arr                         // => [2, 3, 4, 5, 6]
```

异常及错误处理

```
try{
  可能发生异常的代码
} catch(e) {
  这段代码只在异常产生时执行 // e 引用了 error对象
} finally {
  无论是否有异常产生，这段代码都会执行
}
```

你可以通过raise exceptionObject来抛出自定义的异常（可以是任何类型的对象）。

数学函数

全局的Math对象中包含了各种常用的操作以及一些有用的常量，举例如下。

- 幂相关：E LN2 LN10 LOG2E LOG10E SQRT1_2 SQRT2 exp log pow sqrt
- 三角：PI acos asin atan atan2 cos sin tan
- 取整及其他：abs ceil floor max min random round

异步代码：使用timeout

```
var timer;
function callback(){
```

```
    timer && window.clearTimeout(timer);
    timer = null;
    回调代码;
}

window.setTimeout(callback, 100); //0.1秒
```

可怜人的调试器[①]

全局的window对象含有alert(...)方法，调用它可以弹出一个对话框。如果可能的话（使用Safari、装有Firebug的Firefox以及Opera、IE8+），使用控制台机制来获得更轻松的调试体验。

Firebug的console对象在console.log的基础上增加了大量的内容。为了获得更多信息，请登录 http://getfirebug.com/logging。

循环

```
for(decl; test; incr){
  代码
}

do {
  至少会执行一次的代码
} while(条件);
```

```
while(cond) {
  可能永远不会执行的代码
}

while(true) {
  至少会执行一次的代码
  if(跳过接下来的内容)
    continue;
  if(停止当前循环)
    break;
其余的代码
```

“快速数组循环”

```
for(var i=0, l=arr.length; i<l; ++i)
  代码
```

非常有用的字符串操作

这里并没有提到length属性（返回包含的字符个数，而非字节的个数）。

```
'yay'.charAt(1)                        // => 'a'
'hello'.indexOf('l')                   // => 2
'hello'.indexOf('l', 3)                // => 3（指定了搜索起始位置以最小返回值）
'hello'.lastIndexOf('l')               // => 3
'hello'.lastIndexOf('l', 2)            // => 2（指定了搜索结束位置以最大返回值）
'hello'.replace('l', 'L')              // => 'heLlo'
'hello'.replace(/[aeiouy]/g, '-')      // => 'h-ll-'
'hello'.replace(/(.)\1/g, function(s){
```

① 本书作者认为使用alert(...)进行JavaScript调试过于原始，阅读附录B以获得关于JavaScript调试的更多细节。*

```
  return s.toUpperCase();
})                    // => 'heLLo'

'a b c'.split(' ')              // => ['a', 'b', 'c']
'a b \nc'.split(/\s+/)          // => ['a', 'b', 'c']
'abc'.split('')                 // => ['a', 'b', 'c']
'a b c'.split(' ', 2)           // => ['a', 'bc'] (指定了最多分割为多少个)
'hello'.substring(2)            // => 'llo'
'hello'.substring(2, 4)         // => 'll' (指定了子串结束位置)
'hello'.slice(-3)               // => 'llo' (从倒数第3个字符开始)
'hello'.slice(-3, -1)           // => 'll' (指定了结束位置，到倒数第2个字符)
'ÉlodiE'.toLowerCase()          // => 'élodie'
'déjà ça'.toUpperCase()         // => 'DÉJÀ ÇA'
'hello'.match(/(.)\1/)          // ['ll', 'l'] †
```

- 下标0处为完整匹配，下标1处为组1捕获的内容，依此类推。

一些全局的好东西

```
var x = 4;
eval('x * Math.sqrt(16)')            // => 20 - 请小心使用!
decodeURIComponent('%C3%89lodie')    // => 'Élodie'
encodeURIComponent('S & H')          // => 'S%20%26%20H'
parseInt('010')                      // => 8 †
parseInt('08')                       // => NaN †
parseInt('010', 10)                  // => 10 ††
parseFloat('314.15e-2')              // => 3.1415
isNaN(parseInt('foobar'))            // => true
Math.PI.toFixed(3)                   // => 3.142
(5.31762).toFixed(3)                 // => 5.318
Math.PI.toPrecision(3)               // => 3.14
```

- 如果没有显式声明基数，`parseInt`会自动作出判断。前缀为0代表八进制，因此……
- 如果显式声明基数的话，就不会出现混乱了。推荐使用这种形式调用`parseInt`。

标识符和保留字

标识符的起始字符必须是`$`、`_`或者Unicode字符（即使是这样，你仍应避免在标识符中包含怪异的字符），接下来的字符可以是任意Unicode字符。

JavaScript现在的保留字不超过56个，而且它们中的大多数并不是当前版本JavaScript中真正的关键字。

abstract boolean break byte case catch char class const continue debugger default delete do double else enum export extends final finally float for function goto if implements import in instanceof int interface long native new package private protected public return short static super switch synchronized this throw throws transient try typeof var void volatile while with

附录B　JavaScript调试指南

B.1　此处危险

就在几年前，调试JavaScript就像在蹦床上一边跳着一边蒙着眼睛使用硝化甘油[1]一样。那时候，JavaScript的开发工具极度匮乏，而仅有的那几个工具既不好用也不实用（不过平心而论，当时Visual Web Developer Express里面的IE调试工具算是相当不错了）。

总之，在那时候进行JavaScript调试可是一项相当危险的工作。出于Mozilla多年使用捉鬼敢死队中的角色进行开发代号命名的传统，Mozilla的JavaScript调试器被命名为Venkman，而它的底线就是"别死了都不知道是怎么死的!"[2]没错，JavaScript调试在当时就是这样险象环生。

随后，Joe Hewitt进入JavaScript界，推出了他谦称为"只是把一些脚本放在了一起"的工具[3]。事实上，Firebug在Web前端开发这块被遗弃的领域上树立起了一座灯塔，拯救无数JavaScript开发者于水深火热之中。

Firebug的出现把JavaScript开发工具推到了全新的高度，游戏规则也因此发生了巨大的变化，来自各个浏览器供应商的优秀开发者纷纷开始在JavaScript上大展身手，Safari的Web Inspector和Opera的Dragonfly应运而生，甚至Internet Explorer 8都拥有了相当不错的JavaScript调试器。不过，Firebug仍然在JavaScript调试器中保持着领先的地位（尽管领先优势不像以前那么明显），而它也在Web开发者心中占据了一个特殊的位置。

配置一个调试工作台

这篇附录通篇使用了一个带有一些JavaScript脚本的测试页面，以便演示不同浏览器中的调试机

① TNT炸药的主要成分。*
② 电影捉鬼敢死队中的经典台词，原话为"Don't cross the streams!"。*
③ 指Firebug。*

制。该页面的源代码debugbench.html和对应的debugbench.js文件可以在本书的在线代码档案中找到。

下面是这个HTML页面的源代码：

debugbench.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <script type="text/javascript" src="debugbench.js"></script>
  <title>Pocket JavaScript debug bench</title>
</head>
<body>
  <h1>Debug bench</h1>
  <input type="button" onclick="alert(fibo(20))" value="Fibo(20)" />
</body>
</html>
```

下面是接下来会用到的JavaScript源代码：

debugbench.js

```
function fibo(base) {
  if (base <= 2)
    return 1;
  return fibo(base - 1) + fibo(base - 2);
}
```

B.2 Firefox和Firebug

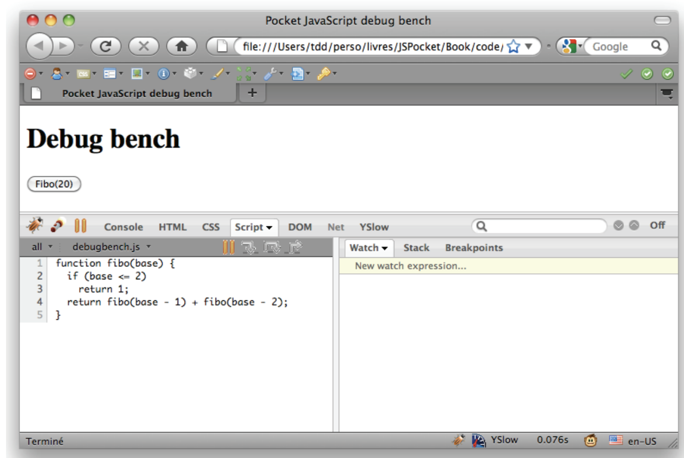
可以在Firebug的官方主页（http://getfirebug.com）或Mozilla的附件库页面（http://addons.mozilla.org）获得Firebug。在写这本书的时候，Firebug已经发布了1.4版本[①]，它的alpha版本也在紧锣密鼓的测试之中。Firebug曾经有过一段比较艰难的时期，当时它的一些Ajax相关的特性会导致双重请求之类的怪异行为。不过经过开发者的努力，在Firebug的新版本里，这些问题已不复存在，同时它的功能也变得更加强大。所以，如果你还在使用Firebug 1.2到1.3之间的版本，那么我建议你应该尽快升级到最新版本。

出于性能方面的考虑，在浏览任何地址（包括本地文件）时，Firebug的一些特性默认处于关闭状态。这时，你需要单击状态条右边的bug图标来打开Firebug面板，并激活你需要的特性。

Firebug的功能非常丰富，不过，本书主要关注Console（控制台）和Script（脚本）选项卡。

① 翻译本书时Firebug已经推出了1.7版本。*

控制台既是一个即时的JavaScript命令行，也是一个日志区。你可以在控制台里利用全局的console对象和内置的方法来编写脚本（内置方法有`log()`、`debug()`、`error()`和`group()`等。如需了解更多，请到Firebug官方站点阅读完整的`console` API[①]）。



图B-1　Firebug的基本Script视图

Script选项卡内包含实际的调试器。它允许你浏览当前页面中所有的脚本（包括加载的文件、内联事件属性等）、设置断点、单步调试代码、监视指定的表达式或变量（Watch子选项卡）以及观察当前栈轨迹（Stack子选项卡）。这正是我们在运行时检查JavaScript代码所必需的。

图B-1展示了在我们的测试页面上运行脚本调试器的效果。

在脚本调试器中运行代码主要有两种方法：进入控制台，执行可以触发待检查代码段的代码；或者在页面里触发待检查代码所对应的事件[②]。在我们的调试测试页面中，单击按钮后，`fibo()`会被调用（参数是20）。你可以自己动手确认——你应该会看到一个显示（6,765）的弹窗。为了理解函数的执行细节，你还可以在调试器中做一些其他操作。

- 选择一个脚本文件（或者是带有内联脚本的HTML页）：单击源码面板上面的文件名旁边的箭头，然后选择你需要的文件。
- 在指定代码行增加及删除断点：单击源码面板指定行号左边的沟槽处即可。
- 设置一个在给定条件下才会发生的断点：右击沟槽处的断点，然后输入JavaScript条件表达式。

① 详见http://getfirebug.com/wiki/index.php/Console_API。*

② 大多数调试器都会在未捕获异常抛出时自动进入单步调试状态。在Firebug中，单击Script选项卡上的箭头就可以启动这个功能。

- 禁用一个断点（同时保留它的位置和条件）：在Breakpoints子选项卡上反选指定断点上的复选框。
- 当一个断点被触发时单步执行代码：源码面板上面的图标允许你单步进入（如果当前的代码行调用了一个函数，执行这个操作会单步进入这个函数内部）、单步执行（直接执行函数调用，而不进入函数内部）和单步跳出（执行当前所在的函数，然后从当前的函数跳出到调用函数位置）。
- 即时观察局部变量和你感兴趣的表达式的值：在Watch子选项卡中完成这些操作。

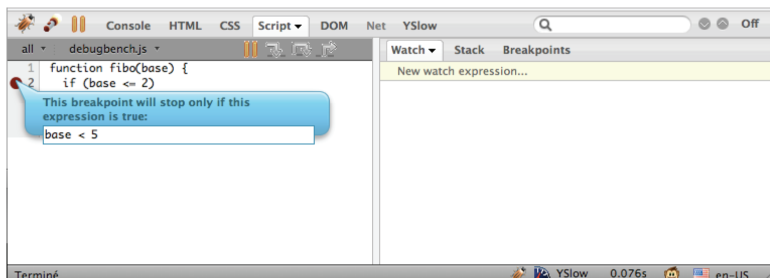
为了让你熟悉上面的操作，我在这里给出一个例子，在这个例子中会使用到上面提到的所有操作，当然你也可以在其他浏览器中尝试这个例子。首先，假设我们想在`fibo()`函数被调用且传入的参数足够小（比如，小于5）的情况下进入单步模式。为了做到这一点，我们首先需要在这个函数的第一行设置断点，然后在这个断点上配置合适的条件。

(1) 在函数的首行（对应图中的第2行）加上断点。

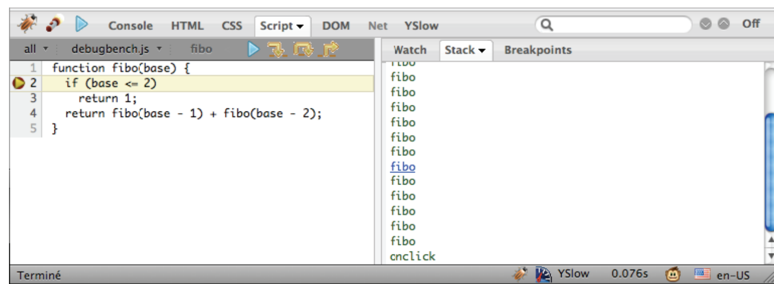
(2) 右击刚刚加上的断点，然后在提示框中输入你需要的条件：`base<5`。确认你所看到的和图B-2中的情形相一致。

一切都设置好了！现在单击Web页面中的Fibo(20)按钮，然后函数会开始递归的执行，直到断点被触发。单击右边面板中的Stack子选项卡来查看你现在所在的位置：你应该会看到类似图B-3中的画面，这些内容展示了当前的递归状态：已经嵌套了15层（如果向下滑动滚动条，你会看到初始的`onclick`事件处理器调用）。

单击Watch子选项卡，你可以看到所有的局部变量，包括当前绑定（**`this`**对象当前所引用的内容）和参数。在图B-4中，你可以看到我们正在执行一个全局函数（因为**`this`**正指向全局的`window`对象），`base`参数的值为4。注意你也可以在Watch子选项卡里修改这些值：双击列表中欲修改的值，然后键入新值。

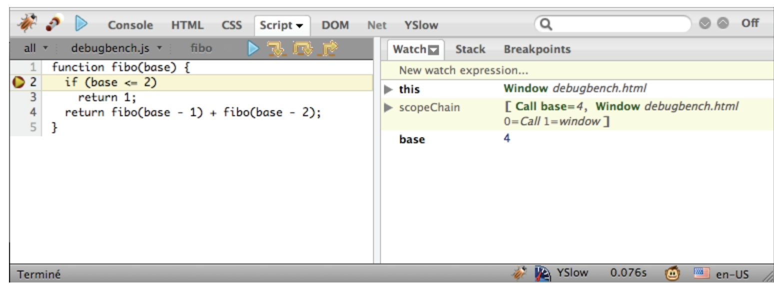


图B-2 在Firebug中设置一个有条件的断点



图B-3　在Firebug中观察栈轨迹

接下来让我们看看不同的单步操作。你现在应该位于`fibo()`函数内部的第1行（也就是代码的第2行）。单击位于蓝色的Play图标（用来恢复执行）右边的Step Into图标，由于条件为假，下面一行被跳过，然后到达**return**所在的这一行。再次单击这个按钮，由于这一行有函数调用，而你使用的是单步进入，因此你会递归地进入同样的函数，而此时`base`值为3（请在Watch子选项卡中加以确认）。



图B-4　Firebug中的Watch视图

假如你想回到你离开时的那一层（`base`值等于4），而不想再一层又一层地进入函数内部，你可以依照下列步骤。

(1) 删除或禁用之前你设置的断点。否则每当用小于5的参数调用`fibo()`时，断点就会被触发。单击Breakpoints子选项卡，反选这个断点以禁用它。

(2) 单击 Step Out 图标（源码面板上部的最后一个图标），让代码继续执行，直到回到之前的栈轨迹位置。这时你会发现自己位于之前的**return**行处，`base`值也恢复为原来的4。

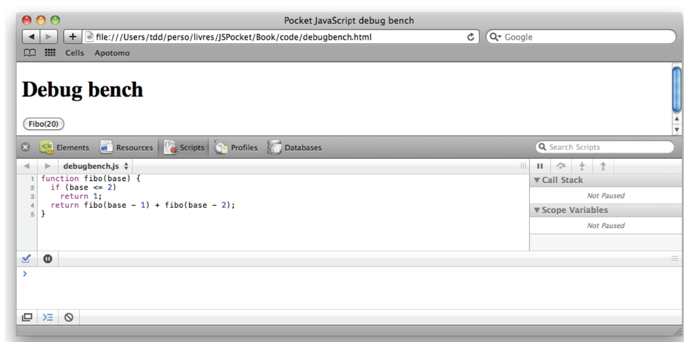
(3) 如果你重复上面的操作，你会一层又一层地回退（也意味着`base`值会回溯到越来越高的值），如此这般操作，过一会儿你会发现Stack子选项卡里面的栈轨迹减少了很多。

(4) 如果你厌倦了手动的单步执行，而想恢复正常执行，直接单击 Play 按钮（带有蓝色箭头的按钮）即可。

需要注意，所有的单步或执行图标都有对应的快捷键，但是这些快捷键会根据你所使用的平台而有所不同。由于图标的tooltip一般会显示该图标对应的快捷键，所以你只需把鼠标指针停留在图标上，然后查看这些快捷键。

B.3 Safari和Web Inspector

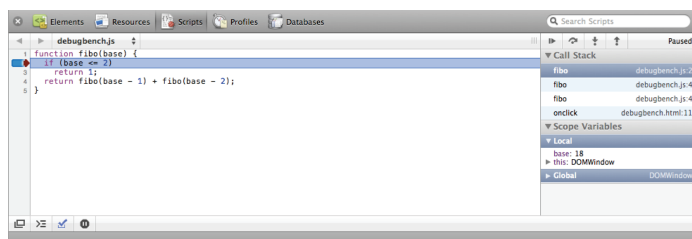
这些年Safari一直在改进Web Inspector，在4.0版本中，Web Inspector已经具备了相当强大的功能（听说最新的版本已经取得了相当的进展）。虽然Web Inspector的DOM修改功能比同类软件稍弱（和Firebug相比），不过它的控制台和JavaScript调试器已经和Firebug旗鼓相当。



图B-5 Safari的Web Inspector

图B-5展示了控制台面板开启的Web Inspector的Scripts选项卡（不管当前的选项卡是什么，你都可以单击底部的图标或按Esc键来切换控制台的开闭状态）。

在当前Safari的公共发行版中，Web Inspector和Firebug在一些关键部分有所不同。比如，你不能删除一个断点，你只能禁用这个断点；你也无法在断点上设置条件，而且也没有监视功能。尽管如此，Web Inspector仍然是一个很有用的工具（而且其后续的版本会越来越好）。图B-6展示了Web Inspector的递归调用栈显示功能，可以看出，在这方面它和Firebug很相似。



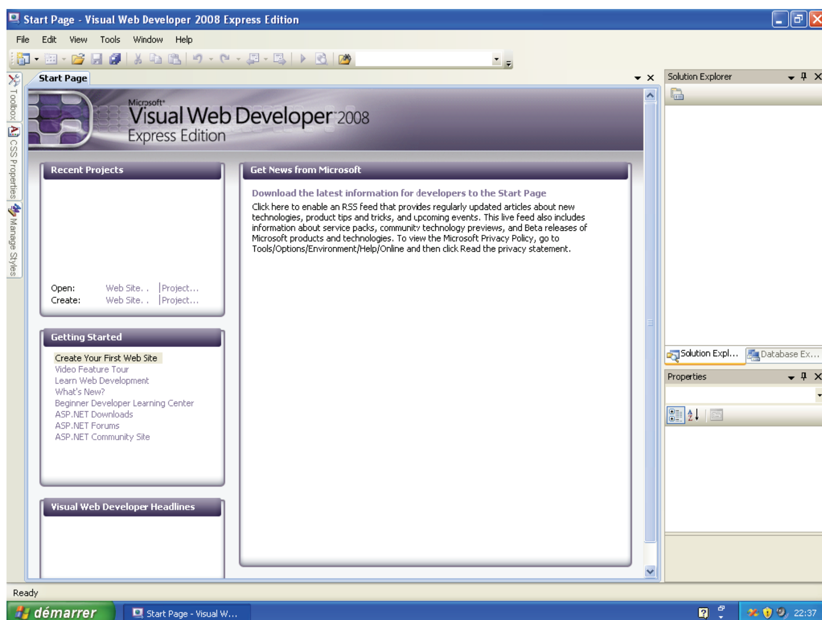
图B-6 在Web Inspector中进行单步调试

B.4　IE6、IE7、IE工具栏以及Web开发者工具

Internet Explorer一直被视为Web开发者的克星，这是因为浏览器之战以网景战死沙场而结束之后，IE的开发突然陷入停滞状态。好在微软在最近的IE版本（IE8和IE9已经取得了非常大的进步）中投入了巨大的人力和财力，不过先前的IE版本（IE6和IE7）在Web标准面前已经落后了不止十年，对JavaScript的支持也是一样。

雪上加霜的是，IE6和IE7中连一个好用的JavaScript调试器都没有。它们只有一个错误百出的控制台，而这就是全部。如果你在这几个版本的IE中寻求调试方案，那么下面这几个选择值得你考虑。

抛开过时的Microsoft Script Debugger不谈，最常用的工具是Visual Web 开发者便捷版：它是微软的大型工具Visual Web开发者工具（实际上是一个定制版的Visual Studio，主要面向Web开发者）的免费受限版本。它很早就发行了，而且它有一个相当不错的JavaScript调试器。



图B-7　Visual Web开发者工具2008快捷版的欢迎页面

所以，除非你正在使用这些工具，否则你就得下载一个多达数G的巨大文件[①]（那个2.6MB的安装文件只是一个代理工具），而这只是为了调试JavaScript。但如果你确实需要在IE6或IE7上

① 下载地址http://www.microsoft.com/express/download/default.aspx。

调试JavaScript，这是值得的，除非你想被这两个浏览器搞崩溃。注意Silverlight和SQL Server都不是必需的安装配置项，因为我们只需要能够正常调试JavaScript就可以了。

安装完成之后，启动Visual Web开发者便捷版（由于这一串词实在太拗口，所以接下来我用“VWDE”来代表它）。在配置完初始项之后，你会看到类似于图B-7的欢迎页面。

由于Express版是免费的，所以它有一些功能限制，比如它不允许把源代码附加到一个运行的浏览器上。不过，你可以使用一个技巧：利用VWDE生成附加浏览器。这个浏览器是VWDE在调试模式下生成的，所以一旦出现JavaScript错误，VWDE的调试器就会把这个错误拦截下来。

我们可以用附加浏览器浏览我们所需要的任何东西。一旦出现错误，VWDE会自动进入调试器界面，我们既可以在这里获取运行中的文档，也可以在其中添加断点。你需要后者来执行本篇附录第一节的操作。

为了生成附加的IE6或IE7浏览器，我们需要对VWDE的需求作出一些妥协。

下面是我们必须遵守的两条原则。

- 我们必须建立一个开启调试模式的VWDE工程。
- 我们必须把IE7设置为默认浏览器，以确保在使用“Start debugging”功能时启动它。

这两个需求都只需设置一次（除非你在这台机器上做测试，以至于需要不停地切换默认浏览器）。下面是具体的操作流程。

(1) 打开IE6或IE7，然后单击工具 > Internet选项……

(2) 进入高级选项卡，然后滑动到浏览区。

(3) 勾选“脚本运行出错时提示”，同时反选“禁用脚本调试……”。

(4) 关闭IE。

(5) 单击“文件” > 新建网站……

(6) 选择“空网站”选项，设置好网站所在的目录后，单击“确认”。

(7) 单击“开始调试”菜单项或对应的图标（带有绿色箭头的图标）。

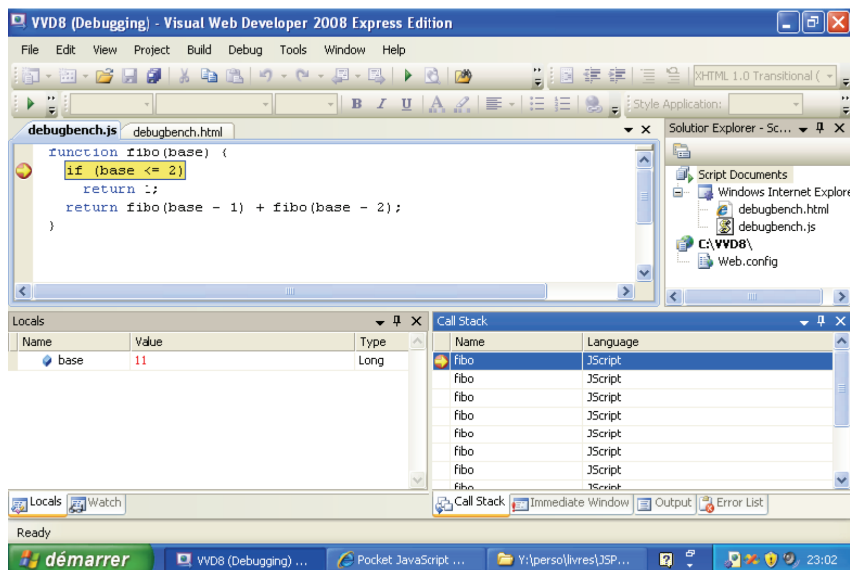
(8) 在第一次启动的时候，VWDE会检测到你的项目缺少一个特殊的Web.config文件（调试所需的配置文件）。它会弹出对话框询问你是否要在关闭调试的状态下继续——很明显，根据提示勾选第一个选项，然后单击确认。

(9) 接下来你会发现任务栏中多了一个服务器图标，这代表一个绑定在动态端口的开发服务器已经启动。接着，IE打开了一个以项目目录为名称的空目录清单（这里可能会提示一个警告，显示浏览器当前在Intranet安全区域下工作，不过这对我们的工作没有影响）。到这里，一切已配

置完毕，准备开始吧！

现在我们已经完成了所有配置，接下来打开debugbench.html文件，然后进入VWDE。在右边的解决方案浏览器面板内，你会看到里面列出了正在运行的两个文件：debugbench.html和debugbench.js，双击后者。

这时，你就可以在源码窗口里按照之前的方式设置断点了（单击沟槽区），设置完成后，切换回IE，然后单击Fibo(20)按钮，你就会进入调试器中的单步调试模式，如图B-8所示。请看！



图B-8　调试IE7里的脚本

B.5　IE8 和开发者工具

毫无疑问，Internet Explorer 8在IE世界中迈出了一大步。除了支持Web标准（虽然和其他的浏览器仍有很大差距，但它总算脱离了史前时代），作为JavaScript开发者的你肯定会对下面这些内容感兴趣。

- IE8中的JScript引擎的速度是其他浏览器中的JavaScript引擎的七分之一左右。它和Chrome的V8、Safari最新的Squirrelfish Extreme以及Firefox的Tracemonkey差距尤其明显——前者的速度是后者的十二分之一左右。

  这听起来一点也不像是什么好消息，但要注意，上一个版本的IE7中的JScript引擎的速度为IE8中的九十分之一左右，更不用说IE6了。IE团队把JScript引擎的速度提高了大约13倍，这已

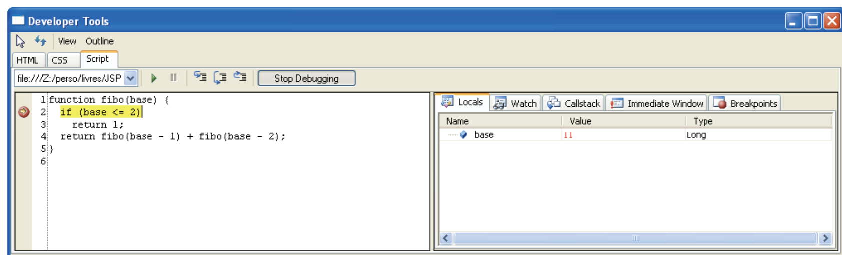
经相当不错了，喜不喜欢由你。

- IE8中的DOM性能（曾经是JavaScript开发者编写脚本的噩梦）得到了显著的提升——现在“只”是其他浏览器的四分之一，相对之前版本的IE，这已经是很大的改进了。
- 相对于IE6和IE7中糟糕的开发者工具栏，IE8内置的“开发者工具”要强大整洁得多，它内置一个真正的JavaScript调试器——一个不需要在IE和外部调试工具间来回切换的JavaScript调试器。

所以，现在我们不但可以在IE世界中实现那些绚丽的效果和Ajax操作，而且还能够更加轻松地完成调试工作！

顺便提一下，尽管IE8做了不少工作，以减少脚本载入造成的可察觉的性能影响，你仍应该使用一些通用的优化方式（比如底部载入脚本[1]、脚本拼接和gzip’ping[2]）。

图B-9展示了处于调试状态中的IE8开发者工具面板，在脚本选项卡中，你可以看到熟悉的断点。单击工具条上的大按钮就可以轻松地切换调试器的开启/关闭状态。单步调试选项、本地变量、监视、调用栈检查、控制台（我真希望微软不要再管它叫“即时窗口”[3]……）等功能一应俱全，断点可以被禁用或删除（不过现在还不能在断点上设置条件）。



图B-9 IE8 开发者工具：Script选项卡

这正是我们所需要的！这套工具用起来还不错，至少到目前为止，它还没有把我的IE8搞崩溃过。

① 参见http://developer.yahoo.com/performance/rules.html#JavaScript_bottom。

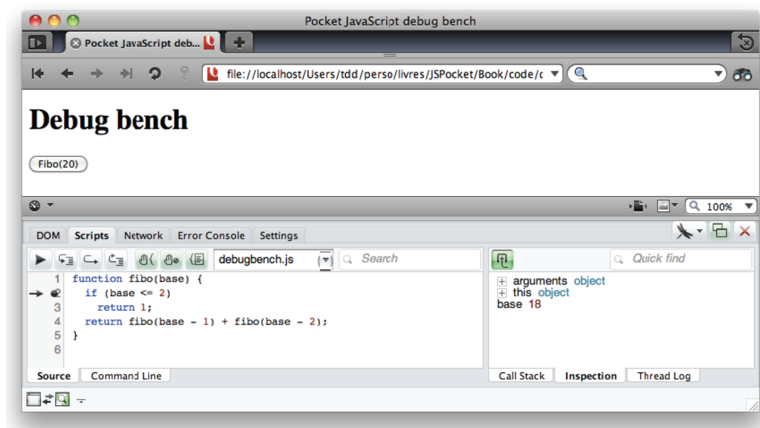
② 如果需要更多关于提升网页载入和渲染性能的资料，请参考Thomas Fuchs和Amy Hoy的*JavaScript Performance Rocks!*一书，或参考http://developer.yahoo.com/performance/rules.html。

③ 微软在IE9中已把即时窗口更名为“控制台”。*

B.6　Opera和Dragonfly

Opera采用了一种不常见的方法来提供开发者工具。它的Dragonfly采用了“离线在线混合模式”，由于Opera会在你打开Dragonfly面板的时候连接互联网，然后自动下载更新，这就意味着你在第一次使用它的时候需要处于在线状态，不过这应该不算什么问题。不过，根据我个人的经验，我发现它似乎必需处于在线状态才能够正常工作。在离线状态下，即便我之前下载过它，它在启动时仍然经常由于无法连接Opera的服务器（或其他原因）而被终止。

Dragonfly当前的版本（在本书的写作时还是alpha3）显然还比较粗糙，我希望Opera的设计者能够花时间好好设计一下，而不是搞成现在这样一层套一层的选项卡界面。我认为其他调试工具使用Firebug那样的布局是有原因的（既简单又足够强大），所以Opera实在没必要独辟蹊径，搞出一套既复杂又难用的界面。不过，最近的版本中这个问题似乎得到了改善。



图B-10　Opera Dragonfly：Scripts选项卡

为了实现我们之前的JavaScript调试步骤，我们只需要图B-10中Scripts选项卡里面的内容。工具条上的按钮包含了各种常用的操作：运行/继续、单步调试、单步进入、单步退出以及在何时自动触发调试模式（比如，进入一个函数或碰到一个错误）。你可以在下拉列表中选择你打算进行调试的源文件（JavaScript或HTML），源码左边的沟槽用来设置断点（右键单击断点来禁用/重新启用它）；代码窗口下面（没错，就是这个诡异位置）的Command line子选项卡提供了一个类似于控制台的命令行；右边的选项卡显示了调用栈和局部变量（Inspection子选项卡中）。

总体来说，它可以应付常规的调试工作。不过我得承认，我个人的调试过程是这样的：首先在Safari或Firefox的环境下编写代码，然后到另一个浏览器（Firefox或Safari）中测试效果，并作

出需要的调整。然后按照IE8、IE7到IE6的顺序逐步测试，最后，为了保证代码的可用性，我会在Opera和Chrome中做一个快速检查。

随后，我会按照先前的顺序进行XHTML/CSS开发。同HTML标签和样式类似，如果你的脚本可以在Safari、Firefox和IE里面正常运行，那一般它在Chrome和Opera里面就不会出什么问题。所以，我很少在Opera中调试，因为几乎没有这个必要。

最后提一下，Opera中可以开启一个调试菜单，从而直接访问一些函数，这个功能有点像Firefox中的Web开发者工具条（Web Developer Toolbar）扩展或是Safari的开发菜单。你可以在Opera中下载一个配置文件，以开启这个功能。

B.7 虚拟机是你的朋友

现在内存已经很便宜了。大多数的Web开发者已经在配有4GB到8GB内存的机器上进行开发，更有一些幸运的家伙用上了SSD硬盘，硬件的提升给他们带来了更为流畅的体验。在这种情况下，虚拟机开始进入了专业Web开发者的视野。虚拟机软件一般很便宜（有的甚至是免费的），而且有各种各样的选择：Parallels Desktop[①]（运行在OS X、Windows和Linux之上）、VMware Fusion[②]（运行在OS X之上）或VMware Workstation[③]（运行在Windows之上）、Sun提供的Virtual Box[④]，除此之外还有很多选择。

由于虚拟机功能越来越强大，我们可以在不同的虚拟机中设置不同的浏览器环境（IE6到IE9、Safari、Chrome、Firefox、Opera以及其他浏览器的各个版本）以便于进行测试。之所以要这样做，是因为多数浏览器不允许你在单个操作系统下运行该浏览器的多个版本。如果你非要这么做，那么浏览器的行为可能会和单一版本环境下的浏览器行为不一致（比如，在Windows XP上安装第三方IE内核浏览器所带来的问题）。虚拟机允许你把不同版本的浏览器隔离到独立的OS镜像中。更妙的是，虚拟机允许你在同一台机器上运行这些镜像。一般来说，我会选择OS X作为宿主OS（host OS），这个选择涉及许多因素，但最主要的一点是：在虚拟机中搭建Linux或Windows环境很容易，但在非Mac平台上的虚拟机中搭建OS X环境异常困难。

所以，请务必使用虚拟机！这比使用多台机器，或是使用冲突百出的远程桌面[⑤]要方便很多。

---

① 参见http://www.parallels.com/products/desktop/。
② 参见http://www.vmware.com/fr/products/fusion/。
③ 参见http://www.vmware.com/products/workstation/。
④ 参见http://www.virtualb6ox.org/。
⑤ 现实的开发中，经常会有多个开发者访问同一台机器的远程桌面，造成各种冲突。*

B.8 网络可能是你的敌人

有时，你可能会碰到一些海森堡式bug[①]：只要你一走开，这些bug就会出现。但当你开始单步执行你的代码，或是试图在你的开发环境中重现这个bug的时候，这些bug又莫名其妙地消失了。为什么会这样呢？

这些bug往往会在高延迟或是容易掉线的网络环境（拨号上网，或是低质量的代理服务器）下运行Ajax程序的时候出现。你可以用一些被称为“减速代理”的软件来重现这种环境。在这方面，我最喜欢的工具（可能也是最有名的工具）是Charles[②]，它基于Java开发（因此它可以在所有主流平台下运行）。它不但可以控制带宽、调整延迟和改变网络行为，而且还可以录制和重放网络会话，并提供详细的HTTP/HTTPS监控。这样的工具真可谓神器！

---

① 参见http://en.wikipedia.org/wiki/Heisenbug，指那些开发者一走开就出现，一调试就没影的诡异bug。*

② 参见http://www.charlesproxy.com/。

附录C　JavaScript框架概览

JavaScript本身是一门伟大的语言。不过，如果要与环境（比如DOM、CSS或`XMLHttpRequest`）交互，即使是最常见的客户端场景，纯粹使用JavaScript就像是用石斧和原木造一座摩天大厦一样。这是由两个因素导致的：原始的DOM接口缺乏合理的高层功能；多数浏览器并不完全遵循Web标准，而是自己搞出了这样那样的标准。

大量的JavaScript框架因此而出现。其中的一些框架已经拥有足够多的用户群，并发展得相当成熟。在本篇附录中，我将为你一一介绍这些优秀的JavaScript框架。

现在要认识到这一点：你应该在手头的项目或任务中尽可能使用框架，甚至是多个框架。因为相对于你自己的代码，这些框架的源码更加稳定，因为它们往往经历过更多测试，具备更完善的文档，并拥有更好的支持。因此相对于手工编码，使用这些框架会节省大量的开发时间。想编码没什么错，不过要务实一些！

选择框架是一项重要的任务，你应该注意下面这几个因素。

- 它已经发行多长时间了？它的代码足够成熟和稳定吗？它得到了足够的测试吗？（换句话说，它有没有配套的测试套件？）
- 它是否存在一个强大的、充满活力的社区？容易获得帮助吗？
- 它是否具备良好的文档（最好是官方文档）？
- 它的API设计能够满足你的个人喜好？如果使用这个框架，你将会写出什么样的代码？
- 这个框架是否面向特定的目标？它是否面向特定的开发者群体？如果是这样，它能满足你的目标或是要求吗？

经常有人把代码库的大小也作为一个评价标准，但实际上你并不需关心它。花大把时间载入JavaScript的日子已经一去不复返了。现在即使是一款庞大的、带有大量注释的代码库，通过应用各种技术，我们也可以闪电般地载入它。这些技术包括注释清除、代码压缩[①]、通过HTTP响应

① 注意不是代码混淆，尽管如此，潜在的调试者绝对不会喜欢被压缩成面目全非的代码。

头改善浏览器缓存、CDN分发[1]等。

主流的框架会使用各种各样的机制，以确保你的页面在短时间内载入，所以就不要在代码大小这种细枝末节的事情上浪费时间了，是吧？

C.1 Prototype、script.aculo.us和Scripty2

Prototype是第一款为大众所熟知的JavaScript框架。2005年2月，Sam Stephenson在37signals公司[2]创建了这个框架以提供一个统一的、易用的API，来进行DOM操作、事件处理和Ajax操作。同年Thomas Fuchs创建了script.aculo.us框架以作为Prototype的补充API，从初期的黄褪技术（Yellow Fade Technique）到现在，它已经发展为集视觉效果、拖放操作以及UI Widget于一身的成熟框架。Scripty2是对script.aculo.us的一个彻底的重写和扩展，现在还处于测试阶段[3]，它提供了丰富的视觉效果。

Prototype的开发团队以五六个志愿者为核心，他们中最著名的要数Sam、Tobie Langel和Andrew Dupont。Prototype使用MIT许可证（基本上来说，它是开源的，可以被到处使用），你可以在GitHub上获得Prototype的完整源代码，Prototype还有一个回馈邮件列表和一个bug报告系统。

围绕着Prototype的发展，一个生态系统逐渐形成，这包括PDoc（一个代码内联文档系统）、Evidence（一款单元测试框架）、Sprockets（一个高级JavaScript处理和拼接工具）以及Scripteka（一个插件库）。Prototype拥有一个活跃的社区，他们的大多活动在Google Groups中进行。介绍Prototype的书也有很多，比如Andrew Dupont的*Practical Prototype and script.aculo.us*以及我的*Prototype and script.aculo.us*。

虽然Prototype被认为是第一款为大众所熟知的JavaScript框架，不过在过去的两年中，它的人气一直在慢慢下滑，很明显，jQuery吸引了更多关注。不过，我认为这两个框架所强调的是不同的（尽管有一定的重叠）需求（它们显然也有着截然不同的代码美学）。

Prototype的架构、代码美学以及内聚的API设计使其非常适合编写健壮的API。它借鉴了很多Ruby的优点（比如Enumerable模块），从而使其易于编写包含大量算法的代码。如果你想利用JavaScript编写优秀的应用，并对这些应用做长时间的扩展和维护，那么Prototype是一个不错的选择。但是如果你并不打算了解JavaScript，而只是需要用一些控件和插件拼凑出一个网站的话，那

---

① CDN即Content Delivery Network，其基本思路是尽可能避开互联网上有可能影响数据传输速度和稳定性的瓶颈和环节，使内容传输得更快、更稳定。*

② 著名的互联网创业公司，流行的Ruby on Rails 框架即出于此。*

③ 当前Scripty的最新版本是2010年9月26日的2.0 Beta。*

Prototype或许不是你的选择。

下面列出了这些库在本书编写时的最新版本以及一些你可能需要的链接。

- Prototype 的官方网站是http://prototypeJavaScript.org/，它当前的版本是1.7。
- script.aculo.us 的官方网站是http://script.aculo.us/，它当前的版本是1.8.3，它的开发已经基本停滞，因为开发者已将注意力转移到了Scripty2。
- Scripty2 的官方网站是http://scripty2.com/，它当前的版本是Alpha release 6。
- PDoc 的官方网站是http://pdoc.org/。
- Sprockets 的官方网站是http://getsprockets.org/。
- Scripteka 的官方网站是http://scripteka.com/。
- 支持邮件列表是http://groups.google.com/group/prototype-scriptaculous。
- 反馈邮件列表是http://groups.google.com/group/prototype-core。咨询问题的话，请使用支持邮件列表！
- Prototype在GitHub上的代码存档在 http://github.com/sstephenson/prototype。
- 如果你需要报告bug，请阅读 http://prototypeJavaScript.org/contribute 里面的向导，按要求仔细完成你的报告，然后将其提交到 https://prototype.lighthouseapp.com/。

C.2 jQuery和jQuery UI

2005年8月，John Resig在使用过Prototype之后，认为Prototype提供的功能很不错，只是API和代码风格不合胃口，所以他开始试图创建另一个框架，这就是jQuery。John现在就职于Mozilla，他是一位和蔼的JavaScript大师，jQuery的开发在他的领导下进展十分顺利。在2006年6月的1.0版本之后，受益于社区的推动，jQuery在接下来的两年发展迅猛。到了2009年9月，jQuery项目正式成立，而1.4版本已在2010年1月的代码狂欢中正式推出（本书写作时，jQuery的版本是1.4.2）。

jQuery的主要目标是降低开发门槛。即使用户不了解JavaScript（甚至不了解编程），在jQuery的帮助下，也可以通过简短的代码来装饰Web页面。它的API为JavaScript的常用操作提供了很多快捷方式，同时也提供了一些插件（它们中的一些已经被归入jQuery UI项目）以处理其他操作。以至于近些年我一直听到人们把jQuery描述成“一种DOM操作的DSL[①]”，或者是“jQuery通过选择器来编码，从而使CSS的使用者也可以操作JavaScript”。尽管这些话并不准确（甚至有些以偏概全），但这些形容还是抓住了jQuery的本质。

jQuery幕后的推动者是它强大的团队。除去插件项目、核心、UI以及网站的那两打开发人员，

① 这里的DSL指的是领域特定语言，详见http://en.wikipedia.org/wiki/Domain-specific_language。*

还有半打人通过会议、活动、会面等各种各样的形式来扩大jQuery社区，这大大推动了jQuery的发展。一些企业也开始资助jQuery，不过这并不影响jQuery的开源性质。jQuery使用GPL/MIT的双重许可证，利用GitHub管理代码，通过Trac来进行bug报告。

jQuery项目和jQuery UI的周围围绕着大量的杂志、会议、会面、用户组以及在线论坛。一些jQuery书已经出版，这包括*jQuery for Dummies*①、*jQuery in Action*②、*jQuery: Novice to Ninja*以及*jQuery Cookbook*。

正如我之前所说，我认为jQuery和Prototype所强调的是不同的需求集合。我并不否认jQuery有很多优点，我也看到jQuery吸引了大量的用户，比如那些只是把JavaScript当成工具来用的用户（他们可不想为做几个日常任务就要学习大量的API）。jQuery社区也使jQuery受益良多，活跃的社区既让用户对jQuery更有信心，也为用户提供了更多的支持。在编写本书的时候，jQuery的版本是1.4.2。

下面是一些你可能需要的链接。

- jQuery 的官方站点：http://jquery.com/。
- jQuery UI 的官方站点：http://jqueryui.com/。
- jQuery的GitHub站点：http://github.com/jquery/jquery。
- Bug跟踪系统：http：//dev.jquery.com/。
- 这个地址中列出了jQuery所有的官方论坛：http://docs.jquery.com/Discussion。

C.3　MooTools

MooTools是由Valerio Proietti在2006年编写的，它像一个手工挑选的特性大杂烩，包含了Prototype的大部分特性、一些视觉效果再加上一些领域特定的工具（cookie管理、用来载入Flash的SWFObject包装器等）。它最初的目标是在保持代码紧凑的前提下，提供传统的、基于类的面向对象编程能力（说实话，在现在Google Ajax API、脚本拼接、gzip等技术大行其道的今天，代码紧凑实在不算什么卖点）。

MooTools项目被分为MooTools 核心（框架本身）和MooTools More（也就是插件库）。

MooTools由一个十三人组成的核心团队开发维护，它是一个开源的项目，使用MIT许可证，它在编写本书时的版本是1.2.4。它的代码寄宿于GitHub，通过Lighthouse进行bug追踪和功能变更，

① 连这样的书都出版了，你可以想象一下jQuery有多么流行（译者补充：for Dummies是国外著名的傻瓜书系列，详见http://en.wikipedia.org/wiki/For_dummies）。

② 此书中文版《jQuery实战》已由人民邮电出版社出版。*

而它的支持则可以在Google Group和一个专门的论坛中获得。

MooTools有一本非常不错的书：*MooTools Essentials*。

现在看起来MooTools在本附录中所提到的框架中人气最低。不过在开发者心中，MooTools仍占有相当重要的地位。

下面列出了一些你可能需要的链接。

- 官方站点：http://mootools.net/。
- 源代码：http://github.com/mootools/mootools-core。
- Bug跟踪和功能变更：https://mootools.lighthouseapp.com/。
- 你可以在Google Group（http://groups.google.com/group/mootools-users）和论坛（http://mooforum.net/）获得支持和项目进展。http://mootorial.com/ 上提供了一些MooTools的资料和教程。

C.4 YUI

Yahoo!用户界面库，一般被称为YUI（据我现在所知，它的发音就是逐字母念出），它是由Yahoo!开发者网络（YDN）所维护的开发者资源中的一部分。这个项目从2005年开始，到2006年年初完成了第一个发布版本。它是一个非常健壮的、模块化的和强大的框架，不过它的入门学习曲线要稍微陡峭一些。“YUI 3”包含了YUI的核心JavaScript功能，此外还有很多额外的功能（不一定和JavaScript相关）位于其他的模块里。YUI被大量应用于Yahoo!自己的网站和在线服务中，这也是其稳定性和高性能的一个实证。

要知道YUI的版本2和版本3之间，无论是初始化框架的方式，还是访问模块和特性的方法，都发生了巨大的变化，因此要小心那些可能过时的文档和教程。一般以YAHOO开头的调用都是YUI 2的代码。不过，从YUI 3.1开始你可以集成那些老式的解决方案，从而使编码更加轻松。

虽然YUI的核心并不算是开源的（尽管它有一个BSD许可证，但实际上这部分代码只有YDN才有权提交），不过YUI Gallery允许人们去提交自己的模块。你也可以进行bug报告和功能请求。

大多数的YUI文档不是在YUI框架的官方站点上，就是在YUI库的站点上，因此它具有很强的社区导向性。YDN的布道者Christian Heilmann编写了一系列高质量的文档，这些文档遍布于流行的Web开发在线杂志，包含了上百个循序渐进的示例。这些文档的质量如此之高，以至于YUI的书都显得多余。我能找到的最好的YUI的书还得追溯到YUI2：*Learning theYahoo! User Interface Library*。

YUI现在相当流行。事实上，YUI在那些富互联网应用中使用的最多，因为对那些不要求UI的操作来说，YUI显得过于重量级了，Dojo也是如此。从这方面来说，YUI经常和“reset”和“grid”此类YDN里的资源[①]被一起使用。YUI社区的活动以官方的YUI库网站为中心。

YUI有一些相当不错的“卖点”：

- 它的API内聚性相当好（整个YUI的风格都是一致的）；
- 它得到了大量的测试，并且拥有大量实用的文档；
- 它对可访问性（比如说，对ARIA的支持）作出了特别的支持；
- 它的模块化、按需载入的理念，再加上其背后强大的CDN，使得它的可用性相当强（即便是在轻量级的环境下）；
- 它由一个大型公司（Yahoo!）维护和支持，因此选择它的技术风险要低得多。

在满足需求的前提下，我认为你应该优先选择那些轻量级的库。这样你可以避免学习和使用那些不需要的东西，与此同时，维护和扩展也会变得更加容易。

下面是你需要的四个链接。

- 官方站点：http://developer.yahoo.com/yui/3/。
- 源码库：http://github.com/yui。
- Bug报告和反馈：http://yuilibrary.com/projects/yui3/report。
- 社区资源：http://yuilibrary.com/。

C.5 ExtJS

ExtJS是一套成熟的RIA[②]框架，它包含大量华丽的、桌面风格的UI Widget，比如树形视图（tree view）、数据表格（datagrid）和对话窗口。Jack Slocum在2006年（我是这么认为的）开始开发它，最初它是作为Prototype、jQuery或是YUI的第三方插件存在的。到后来，ExtJS逐渐发展成一个以创建华丽的、桌面风格的用户界面为目标的大型独立框架；比如，它很早就配备了优秀的数据表格、树形视图以及Ajax功能。因此，很多人选择ExtJS用来构建网站管理页面。

ExtJS是与其同名的公司旗下的产品，该公司也提供其他相关的工具：Ext Designer和Ext GWT bridge。ExtJS同时具有三个许可证：GPLv3、OEM和商业许可证。ExtJS许可证的混乱曾在其发行时引起过一阵骚动，因为GPLv3使得它不能被任何非GPL开源项目使用。另一方面，你可以从

① 详见http://developer.yahoo.com/yui/reset/和http://developer.yahoo.com/yui/grids/。*
② 这里RIA指的是富互联网应用，详见http://en.wikipedia.org/wiki/Rich_Internet_Application。*

该库的创始者那里得到ExtJS官方的商业技术支持、培训以及订阅。很遗憾，ExtJS的源码库已经被关闭了，现在只能从某个下载页的快照上访问到它。此外我也没有发现ExtJS有什么像样的Bug跟踪系统，总之，除了GPLv3这个选项以外，ExtJS看起来并不怎么开放。

和其他优秀的项目一样，ExtJS背后有一群全职的、积极的员工[①]来推动它的发展。在编写本书时，ExtJS的版本是3.1.1，在2010年它可能还会发布几个新的主版本。

ExtJS的文档相当不错，也便于浏览，就是有些太过简洁了，与此同时，ExtJS的API参考配备了完整的应用实例。此外也有不少不错的ExtJS书，其中最出名的要数2009年出版的*ExtJS 3.0 Cookbook*，以及2010年夏出版的*ExtJS in Action*。

最后，由于ExtJS项目专属于ExtJS公司，因此它的生态系统被局限在官方站点、论坛以及一个基于维基系统的学习中心（提供演示、录像、教程等学习资料）。

下面是你使用ExtJS所不可或缺的链接。

- 官方站点：http://extjs.com/。
- 下载地址：http://www.extjs.com/products/extjs/download.php。
- 官方论坛：http://www.extjs.com/forum/。
- 学习中心：http://www.extjs.com/learn/Main_Page。

C.6 Dojo

看了那么多的JavaScript流行框架，最后在这里介绍一下Dojo。Dojo项目于2004年开始，经过Alex Russell、Dylan Schiemann和David Schontzler的不懈努力，于2007年11月5日发布了1.0正式版本。Dojo项目由Dojo（核心部分）、Dijit（UI Widget）以及DojoX（插件和扩展功能）组成。

Dojo在很多方面上和ExtJS很相似。它是一个主流的RIA框架，并拥有强大的公司作为后盾。Dojo项目现在由Dojo基金会（赞助公司包括IBM、Google、AOL、Thomson Reuters、TIBCO、Zend、Sitepen等）所支持。它当前（编写本书时）的版本是1.5.0。和ExtJS不同的是，Dojo是一个真正的开源项目，它同时具有学术用自由许可证（Academic Free License 2.1）和新版BSD许可证。GitHub、Subversion以及Bazaar上都有Dojo的代码库，同时，它还拥有一个在线Bug跟踪系统。

Dojo和Dojo基金会下的其他项目关系也很密切，这些项目中最有名的要数cometD和DWR，它们很早就发布了，现在仍然被广泛应用于服务器向浏览器的自动推送服务（不要忘了Sizzle、Persevere以及General Interface）。

---

① ExtJS拥有一个六人的管理团队，以及一个人员数量未知的全职开发团队。

各大Web开发会议中都能看到Dojo的身影（T恤衫、短会、小聚以及优秀的作品），同时它还拥有一个活跃的社区和专门的论坛。随着社区力量的强大，Dojo有可能会成为第二个jQuery。Dojo有不少不错的书，其中包括*Mastering Dojo*①、*Dojo: the Definitive Guide*②、*Practical Dojo Projects*以及刚出版不久的*Getting StartED with Dojo*。

Dojo提供了非常优秀的在线文档，其中包含快速使用指南、参考指南以及API文档，此外还有一系列面向任务的“常用解决方案”（例如，“通过数据集创建图表”）。

我个人认为，如果你需要创建符合Web标准的、RIA式的、桌面风格的应用，Dojo是你的首要选择。

作为本篇附录的结束，下面列出了与Dojo相关的重要链接。

- 官方站点：http://www.dojotoolkit.org/。
- 源码地址：http://svn.dojotoolkit.org/src/（可惜不是Git）。
- Bug跟踪和报告系统：http://bugs.dojotoolkit.org/。
- 社区论坛：http://www.dojotoolkit.org/community/。

① 此书中文版《精通Dojo》已由人民邮电出版社出版。*
② 此书中文版《Dojo权威指南》已由机械工业出版社出版。*

附录D　求助指南

我当然希望这本书会对你有用。不过，你仍然需要去自己动手寻找问题的答案，或是寻求他人的指点以确保自己没走错路。问题在于，在哪里可以得到帮助，或是得到他人的指点呢？

D.1　JavaScript 求助指南

即便存在大量优秀的JavaScript框架，这也不能成为你不了解JavaScript的借口。而且有时你确实需要动手编写一些代码，比如说由于一些原因而不能使用你常用的框架（比如移动设备上的性能限制[①]）。

新闻组

还记得在宽带出现前人们都在用什么吗？还记得在Google Group之前人们在用什么吗？还记得Google出现前人们在用什么吗？好吧，也许你在那时还没有开始编代码，那时我们用的是Usenet[②]。它现在仍然陪伴在开发者身边，新一代的开发者已经发现了它的种种好处，比如说共享二进制文件。除此之外，它还拥有我们所知的所有编程语言的新闻组。

你可以在ISP签订的网络服务协议中找到一个称之为NNTP[③]服务器的东西，这就是ISP提供的新闻组接入服务。大多数的新闻聚合器（aggregator）、源阅读器（feed reader）以及电子邮件客户端（包括Thunderbird、Entourage、Outlook以及Outlook Express）都允许你连接一个NTTP服务器，然后订阅你需要的新闻组（根据ISP的不同，可选的新闻组也会有所差异）。

见鬼，这样访问新闻组也太麻烦了，好在Google Group维持了不少实用新闻组的代理。

---

① 在这方面，你可以试试Thomas Fuchs刚刚发布的zepto.js：http://github.com/madrobby/zepto，或者是更快的vapor.js：http://github.com/madrobby/vapor.js。

② 又称新闻讨论组，它是Uses Network的缩写。它是Internet上信息传播的一个重要组成部分，也是Internet上一种高效率的交流方式。*

③ 即网络新闻传输协议，它主要用于阅读和张贴新闻文章到Usenet。*

- `comp.lang.javascript`是JavaScript的核心新闻组（这个地址是COMPuter LANGuage JavaScript的缩写）。它具有相当长的历史，现在，它仍然是JavaScript语言的主要讨论区。在这里你会遇到各种各样的参与者，他们中既有一无所知的新手，也有令人尊敬的大师。
- 你也可以在一些非英语的子新闻组中进行讨论，比如`japan.comp.lang.javascript`、`de.comp.lang.javascript`、`fr.comp.lang.javascript`等。

邮件列表和论坛

现在有大量的JavaScript论坛，根据发帖者水平的不同，这些论坛的质量也参差不齐。要离那些动辄粘贴复制代码的论坛远些，那是论坛风气的一个不良表现。

- Google Group维持了我刚才提到的JavaScript新闻组的代理，你可以先到那里看看[①]。
- Sitepoint是一个相当不错的站点，它们的JavaScript论坛值得一去[②]。
- Webdeveloper站点有各种各样的活动[③]。

总之，优先考虑第一个选择（Google提供的新闻组代理）。那里的帖子质量非常高。

IRC[④]频道

啊，IRC。这是另一个老古董了，这里之所以提到它，是因为它提供了即时交流的功能。另一方面，它的实用程度取决于你登录时有多少个高手在线。

`irc.freenode.net`上的主要频道是`##JavaScript`。它一般都会有300到400人同时在线。如果你多逛逛的话，可能会发现一些不错的非英语频道。

进一步阅读

现在有大量的JavaScript权威书。我已经在参考书目中列出了不少书目，在这里我重点挑出了几本。

- David Flanagan编写的*JavaScript: The Definitive Guide*[⑤]是公认的JavaScript圣经，比较搞笑的是，确实有一本书名叫“JavaScript圣经”，它就是Danny Goodman编写的*JavaScript Bible*[⑥]，由Brendan Eich[⑦]作序（它的第7版就快出版了）。

---

① 参见http://groups.google.com/group/comp.lang.javascript。
② 参见http://www.sitepoint.com/forums/forumdisplay.php?f=15。
③ 参见http://www.webdeveloper.com/forum/forumdisplay.php?forumid=3。
④ IRC是Internet Relay Chat 的英文缩写，详见http://en.wikipedia.org/wiki/IRC。*
⑤ 此书中文版《JavaScript权威指南》，已由机械工业出版社出版。*
⑥ 此书中文版《JavaScript宝典》，已由人民邮电出版社出版。*
⑦ JavaScript语言的创始人。*

- Douglas Crockford（JSON和JSLint的创始人）编写的*JavaScript: The GoodParts*[①]也很值得一看，Doug在此书中不断提醒我们JavaScript中三分之二的内容都不应被使用（笑）。
- jQuery的创始人John Resig最近在Sitepoint上编写了一本看似不错的小册子：*Secrets of the JavaScript Ninja*。
- 接下来，这本书仍然来自Sitepoint，虽然有些过时，但仍然是一本好书，它由多个作者合著而成，这就是*The Art and Science of JavaScript*[②]。
- 非凡的JavaScript传道者Chris Heilmann编写的*Beginning Java-Script with DOM Scripting and Ajax: From Novice to Professional*也是值得一读的佳作。
- 此外，一定不要忘了Thomas Fuchs和Amy Hoy最近合写的一本书：*JavaScript Performance Rocks!*。这本书对如何调优JavaScript性能有着出色的讲解。

D.2 框架的帮助资源

这部分把分散在附录C中的一些在线资源整合在了一起。

Prototype和script.aculo.us

- 官方支持列表：http://groups.google.com/group/prototype-scriptaculous。
- 在线API文档：http://api.prototypejs.org/。
  http://wiki.github.com/madrobby/scriptaculous/。
- IRC频道：Freenode 上的`#prototype`和`#scriptaculous`频道。

参考书：

- *Practical Prototype and script.aculo.us*，由Andrew Dupont编写。
- *Prototype and script.aculo.us*，由本书作者编写。

jQuery

- 官方论坛： http://forum.jquery.com/。
- 教程：http://docs.jquery.com/Tutorials。
- API文档：http://docs.jquery.com/Main_Page。
- IRC频道：Freenode 上的`#jquery`频道。

---

① 此书中文版《JavaScript语言精粹》，已由电子工业出版社出版。*
② 此书中文版《JavaScript艺术与科学》已由电子工业出版社出版。*

参考书：

- *jQuery for Dummies*，由Lynn Beighley编写。
- *jQuery in Action*，由Bear Bibault 和Yehuda Kata编写。
- *jQuery: Novice to Ninja*，由Earle Castledine 和 Craig Sharkie编写。
- *jQuery Cookbook*，由Cody Lindley编写。

MooTools

- 官方支持列表：http://groups.google.com/group/mootools-users。
- 论坛：http://mooforum.net/。
- 教程：http://mootorial.com/。
- API文档：http://mootools.net/docs/core。

参考书：

- *MooTools Essentials*，由Aaron Newton编写。

YUI

- 论坛：http://yuilibrary.com/forum/。
- 文档：http://developer.yahoo.com/yui/3/。
- IRC频道：Freenode 上的#　　频道 （尽管并不是很活跃）。

参考书：

- *Learning the Yahoo! User Interface Library*，由Dan Wellman编写。

ExtJS

- 论坛：http://www.extjs.com/forum/。
- 实例演示：http://www.extjs.com/deploy/dev/examples/。
- API文档：http://www.extjs.com/deploy/dev/docs/。
- 学习中心：http://www.extjs.com/learn/Main_Page。
- IRC频道：Freenode上的`#extjs`频道（尽管并不是很活跃）。

参考书：

- *ExtJS 3.0 Cookbook*，由Jorge Ramon编写。
- *ExtJS in Action*，由Jesus Garcia编写。

Dojo

- 论坛：http://www.dojotoolkit.org/community/。
- API文档和教程：http://www.dojotoolkit.org/documentation。
- IRC频道：Freenode上的`#dojo`频道。

参考书：

- *Mastering Dojo*，由Craig Riecke、Rawld Gill和Alex Russell编写。
- *Dojo: the Definitive Guide*，由Matthew A. Russell编写。
- *Practical Dojo Projects*，由Frank Zammetti编写。

附录D　求助指南

我当然希望这本书会对你有用。不过，你仍然需要去自己动手寻找问题的答案，或是寻求他人的指点以确保自己没走错路。问题在于，在哪里可以得到帮助，或是得到他人的指点呢？

D.1　JavaScript求助指南

即便存在大量优秀的JavaScript框架，这也不能成为你不了解JavaScript的借口。而且有时你确实需要动手编写一些代码，比如说由于一些原因而不能使用你常用的框架（比如移动设备上的性能限制[①]）。

新闻组

还记得在宽带出现前人们都在用什么吗？还记得在Google Group之前人们在用什么吗？还记得Google出现前人们在用什么吗？好吧，也许你在那时还没有开始编代码，那时我们用的是Usenet[②]。它现在仍然陪伴在开发者身边，新一代的开发者已经发现了它的种种好处，比如说共享二进制文件。除此之外，它还拥有我们所知的所有编程语言的新闻组。

你可以在ISP签订的网络服务协议中找到一个称之为NNTP[③]服务器的东西，这就是ISP提供的新闻组接入服务。大多数的新闻聚合器（aggregator）、源阅读器（feed reader）以及电子邮件

---

① 在这方面，你可以试试Thomas Fuchs刚刚发布的zepto.js：http://github.com/madrobby/zepto，或者是更快的vapor.js：http://github.com/madrobby/vapor.js。

② 又称新闻讨论组，它是Uses Network的缩写。它是Internet上信息传播的一个重要组成部分，也是Internet上一种高效率的交流方式。*

③ 即网络新闻传输协议，它主要用于阅读和张贴新闻文章到Usenet。*

客户端（包括Thunderbird、Entourage、Outlook以及Outlook Express）都允许你连接一个NTTP服务器，然后订阅你需要的新闻组（根据ISP的不同，可选的新闻组也会有所差异）。

见鬼，这样访问新闻组也太麻烦了，好在Google Group维持了不少实用新闻组的代理。

- `comp.lang.javascript`是JavaScript的核心新闻组（这个地址是COMPuter LANGuage JavaScript的缩写）。它具有相当长的历史，现在，它仍然是JavaScript语言的主要讨论区。在这里你会遇到各种各样的参与者，他们中既有一无所知的新手，也有令人尊敬的大师。
- 你也可以在一些非英语的子新闻组中进行讨论，比如`japan.comp.lang.javascript`、`de.comp.lang.javascript`、`fr.comp.lang.javascript`等。

邮件列表和论坛

现在有大量的JavaScript论坛，根据发帖者水平的不同，这些论坛的质量也参差不齐。要离那些动辄粘贴复制代码的论坛远些，那是论坛风气的一个不良表现。

- Google Group维持了我刚才提到的JavaScript新闻组的代理，你可以先到那里看看[①]。
- Sitepoint是一个相当不错的站点，它们的JavaScript论坛值得一去[②]。
- Webdeveloper站点有各种各样的活动[①]。

总之，优先考虑第一个选择（Google提供的新闻组代理）。那里的帖子质量非常高。

---

① 参见http://groups.google.com/group/comp.lang.javascript。

② 参见http://www.sitepoint.com/forums/forumdisplay.php?f=15。

IRC[2]频道

啊，IRC。这是另一个老古董了，这里之所以提到它，是因为它提供了即时交流的功能。另一方面，它的实用程度取决于你登录时有多少个高手在线。

`irc.freenode.net`上的主要频道是`##JavaScript`。它一般都会有300到400人同时在线。如果你多逛逛的话，可能会发现一些不错的非英语频道。

进一步阅读

现在有大量的JavaScript权威书。我已经在参考书目中列出了不少书目，在这里我重点挑出了几本。

- David Flanagan编写的*JavaScript: The Definitive Guide*[3]是公认的JavaScript圣经，比较搞笑的是，确实有一本书名叫“ JavaScript圣经”，它就是Danny Goodman编写的*JavaScript Bible*[4]，由Brendan Eich[5]作序（它的第7版就快出版了）。
- Douglas Crockford（JSON和JSLint的创始人）编写的*JavaScript: The GoodParts*[6]也很值得一看，Doug在此书中不断提醒我们JavaScript中三分之二的内容都不应被使用（笑）。

---

① 参见http://www.webdeveloper.com/forum/forumdisplay.php?forumid=3。
② IRC是Internet Relay Chat 的英文缩写，详见http://en.wikipedia.org/wiki/IRC。*
③ 此书中文版《JavaScript权威指南》，已由机械工业出版社出版。*
④ 此书中文版《JavaScript宝典》，已由人民邮电出版社出版。*
⑤ JavaScript语言的创始人。*
⑥ 此书中文版《JavaScript语言精粹》，已由电子工业出版社出版。*

- jQuery的创始人John Resig最近在Sitepoint上编写了一本看似不错的小册子：*Secrets of the JavaScript Ninja*。
- 接下来，这本书仍然来自Sitepoint，虽然有些过时，但仍然是一本好书，它由多个作者合著而成，这就是*The Art and Science of JavaScript*[①]。
- 非凡的JavaScript传道者Chris Heilmann编写的*Beginning Java-Script with DOM Scripting and Ajax: From Novice to Professional*也是值得一读的佳作。
- 此外，一定不要忘了Thomas Fuchs和Amy Hoy最近合写的一本书：*JavaScript Performance Rocks!*。这本书对如何调优JavaScript性能有着出色的讲解。

D.2　框架的帮助资源

这部分把分散在附录C中的一些在线资源整合在了一起。

Prototype和script.aculo.us

- 官方支持列表：http://groups.google.com/group/prototype-scriptaculous。
- 在线API文档：http://api.prototypejs.org/。

http://wiki.github.com/madrobby/scriptaculous/。

- IRC频道：Freenode 上的`#prototype`和`#scriptaculous`频道。

---

① 此书中文版《JavaScript艺术与科学》已由电子工业出版社出版。*

参考书：

- *Practical Prototype and script.aculo.us*，由Andrew Dupont编写。
- *Prototype and script.aculo.us*，由本书作者编写。

jQuery

- 官方论坛： http://forum.jquery.com/。
- 教程：http://docs.jquery.com/Tutorials。
- API文档：http://docs.jquery.com/Main_Page。
- IRC频道：Freenode 上的`#jquery`频道。

参考书：

- *jQuery for Dummies*，由Lynn Beighley编写。
- *jQuery in Action*，由Bear Bibault 和Yehuda Kata编写。
- *jQuery: Novice to Ninja*，由Earle Castledine 和 Craig Sharkie编写。
- *jQuery Cookbook*，由Cody Lindley编写。

MooTools

- 官方支持列表：http://groups.google.com/group/mootools-users。

- 论坛：http://mooforum.net/。
- 教程：http://mootorial.com/。
- API文档：http://mootools.net/docs/core。

参考书：

- *MooTools Essentials*，由Aaron Newton编写。

YUI

- 论坛：http://yuilibrary.com/forum/。
- 文档：http://developer.yahoo.com/yui/3/。
- IRC频道：Freenode 上的#yui频道（尽管并不是很活跃）。

参考书：

- *Learning the Yahoo! User Interface Library*，由Dan Wellman编写。

ExtJS

- 论坛：http://www.extjs.com/forum/。
- 实例演示：http://www.extjs.com/deploy/dev/examples/。
- API文档：http://www.extjs.com/deploy/dev/docs/。

- 学习中心：http://www.extjs.com/learn/Main_Page。
- IRC频道：Freenode上的`#extjs`频道（尽管并不是很活跃）。

参考书：

- *ExtJS 3.0 Cookbook*，由Jorge Ramon编写。
- *ExtJS in Action*，由Jesus Garcia编写。

Dojo

- 论坛：http://www.dojotoolkit.org/community/。
- API文档和教程：http://www.dojotoolkit.org/documentation。
- IRC频道：Freenode上的`#dojo`频道。

参考书：

- *Mastering Dojo*，由Craig Riecke、Rawld Gill和Alex Russell编写。
- *Dojo: the Definitive Guide*，由Matthew A. Russell编写。
- *Practical Dojo Projects*，由Frank Zammetti编写。

参考文献

[AEH+07]] Cameron Adams, James Edwards, Christian Heilmann, Michael Mahemoff, Ara Pehlivanian, Dan Webb, and Simon Willison. *The Art and Science of JavaScript*. Sitepoint, 2007.

[Bei10] Lynn Beighley. *jQuery for Dummies*. For Dummies, 2010.

[BK10] Bear Bibeault and Yehuda Katz. *jQuery in Action*. Manning Publications Co., Greenwich, CT, second edition, 2010.

[Cro08] Douglas Crockford. *JavaScript: The Good Parts*. O'Reilly Media, Inc. / Yahoo! Press, Sebastopol, CA, 2008.

[CS10] Earle Castledine and Craig Sharkie. *jQuery: Novice to Ninja*. Sitepoint, San Francisco, CA, 2010.

[Dup08] Andrew Dupont. *Practical Prototype and script.aculo.us*. Apress, New York, NY, 2008.

[Fla06] David Flanagan. *JavaScript: The Definitive Guide*. O'Reilly Media, Inc., Sebastopol, CA, fifth edition, 2006.

[Gar10] Jesus Garcia. *ExtJS in Action*. Manning Publications Co., Greenwich, CT, 2010.

[Goo07] Danny Goodman. *JavaScript Bible*. John Wiley & Sons, 2007.

[Hay10] Kyle Hayes. *Getting StartED with Dojo*. Friends of ED, New York, NY, 2010.

[Hei06] Christian Heilmann. *Beginning JavaScript with DOM Scripting and Ajax: From Novice to Professional*. Apress, New York, NY, 2006.

[HF09] Amy Hoy and Thomas Fuchs. *JavaScript Performance Rocks!* Slash7, Vienna, Austria, 2009.

[Lin09] Cody Lindley. *jQuery Cookbook*. O'Reilly Media, Inc., Sebastopol, CA, 2009.

[New08] Aaron Newton. *MooTools Essentials*. Apress, 2008.

[Por07] Christophe Porteneuve. *Prototype and script.aculo.us: You never knew JavaScript could do this!* The Pragmatic Programmers, LLC, Raleigh, NC, and Dallas, TX, 2007.

[Ram09] Jorge Ramon. *ExtJS 3.0 Cookbook*. Packt, Birmingham, UK, 2009.

[Res09] John Resig. *Secrets of the JavaScript Ninja*. Manning Publications Co., Greenwich, CT, 2009.

[RGR08] Craig Riecke, Rawld Gill, and Alex Russell. *Mastering Dojo: JavaScript and Ajax Tools for Great Web Experiences*. The Pragmatic Programmers, LLC, Raleigh, NC, and Dallas, TX, 2008.

[Rus08] Matthew A. Russell. *Dojo: the Definitive Guide*. O'Reilly Media, Inc., Sebastopol, CA, 2008.

[Wel08] Dan Wellman. *Learning the Yahoo! User Interface library*. Packt, Birmingham, UK, 2008.

[Zam08] Frank Zammetti. *Practical Dojo Projects*. Apress, New York, NY, 2008.

“真希望我初学JavaScript那会儿就有这本书！本书会让你在现实的JavaScript世界中游刃有余。它既讲述了当今流行的JavaScript库的运行原理，也提供了JavaScript正确实践的好建议和背景资料。作为最优秀的JavaScript开发者之一，作者将他多年的宝贵经验凝聚在本书中，使它成为日常JavaScript开发中的必读参考。”

——Thomas Fuchs，script.aculo.us框架创始人

“这本书包含了一系列当今浏览器中既精妙又实用的JavaScript贴士和技巧，既有表单验证和JSON处理这样的基本技术，也有混搭和几何定位这样的应用实例。如果你想使用JavaScript编写更优秀的Web应用，我强烈建议你阅读本书。”

——Dylan Schiemann，SitePen的CEO，Dojo工具箱联合创始人

Pragmatic Guide to JavaScript

JavaScript修炼之道

JavaScript已无处不在。在当今纷繁复杂的网络世界中，它是不可或缺的组成部分。然而，即便对有经验的开发人员而言，JavaScript的体系都像难以穿越的生态环境系统。为此，本书以有别于一般教程的任务驱动方式来组织，围绕35个必会的关键JavaScript任务进行论述，并针对常见任务提出了一些新的开发方法，再加上本书独特的左页原理右页代码的编排方式，使你在阅读过程中快速地获得提升。

在本书中，作者将教会你基本原理、最便利的工具以及业内最佳实践，同时也能帮你简化编程模型，适应更加复杂的交互需求，提升用户在客户端的浏览体验。

如果你熟悉其他任何语言编程，那么通过本书掌握JavaScript将易如反掌。

- 左页原理右页代码，极速修炼Web开发秘技
- 专注于浏览器端脚本编程，提供快速的解决方案
- 任务驱动，以实战掌握JavaScript全貌

The Pragmatic Programmers

图灵社区：www.ituring.com.cn
反馈／投稿／推荐信箱：contact@turingbook.com
热线：(010)51095186转604

分类建议 计算机／程序设计／JavaScript

人民邮电出版社网址：www.ptpress.com.cn



ISBN 978-7-115-26556-2

定价：29.00元